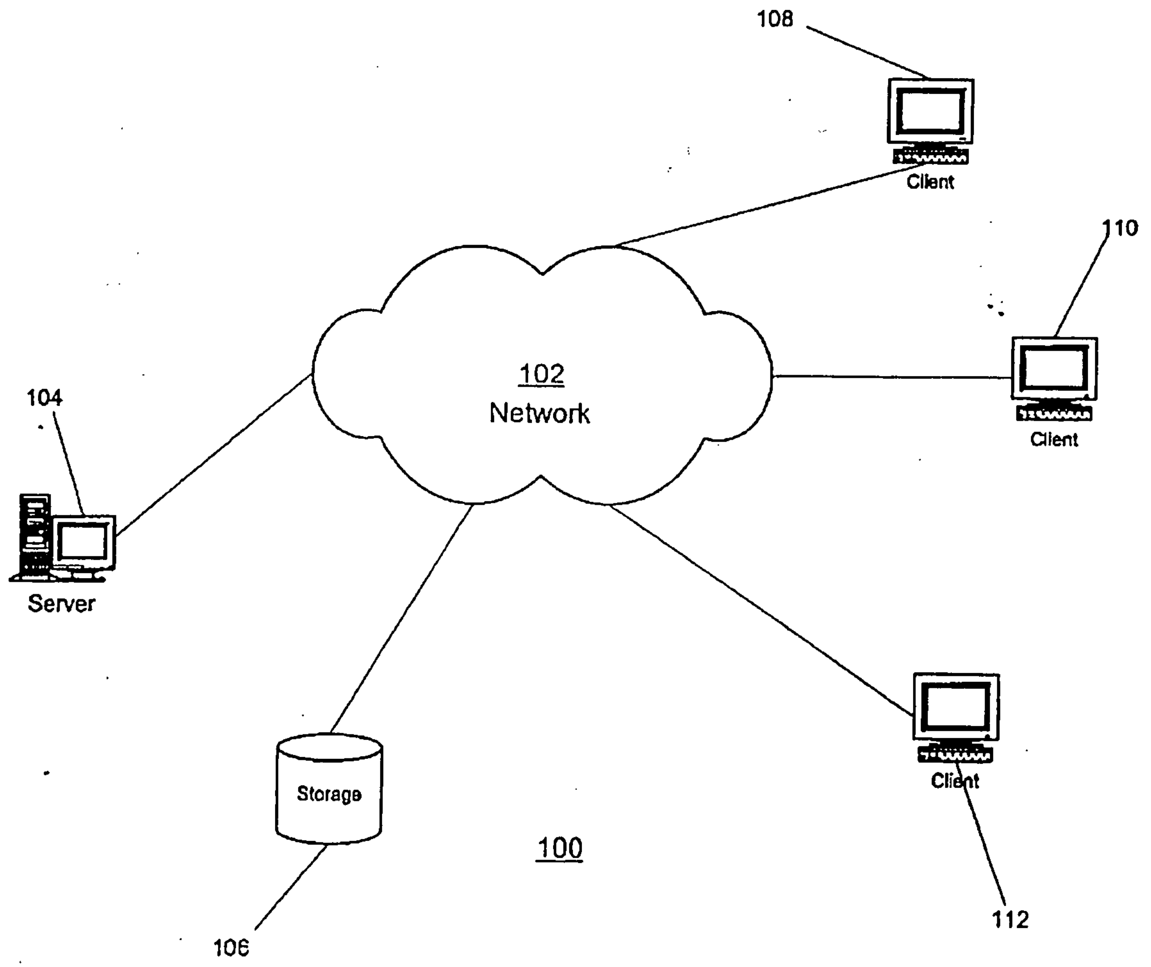
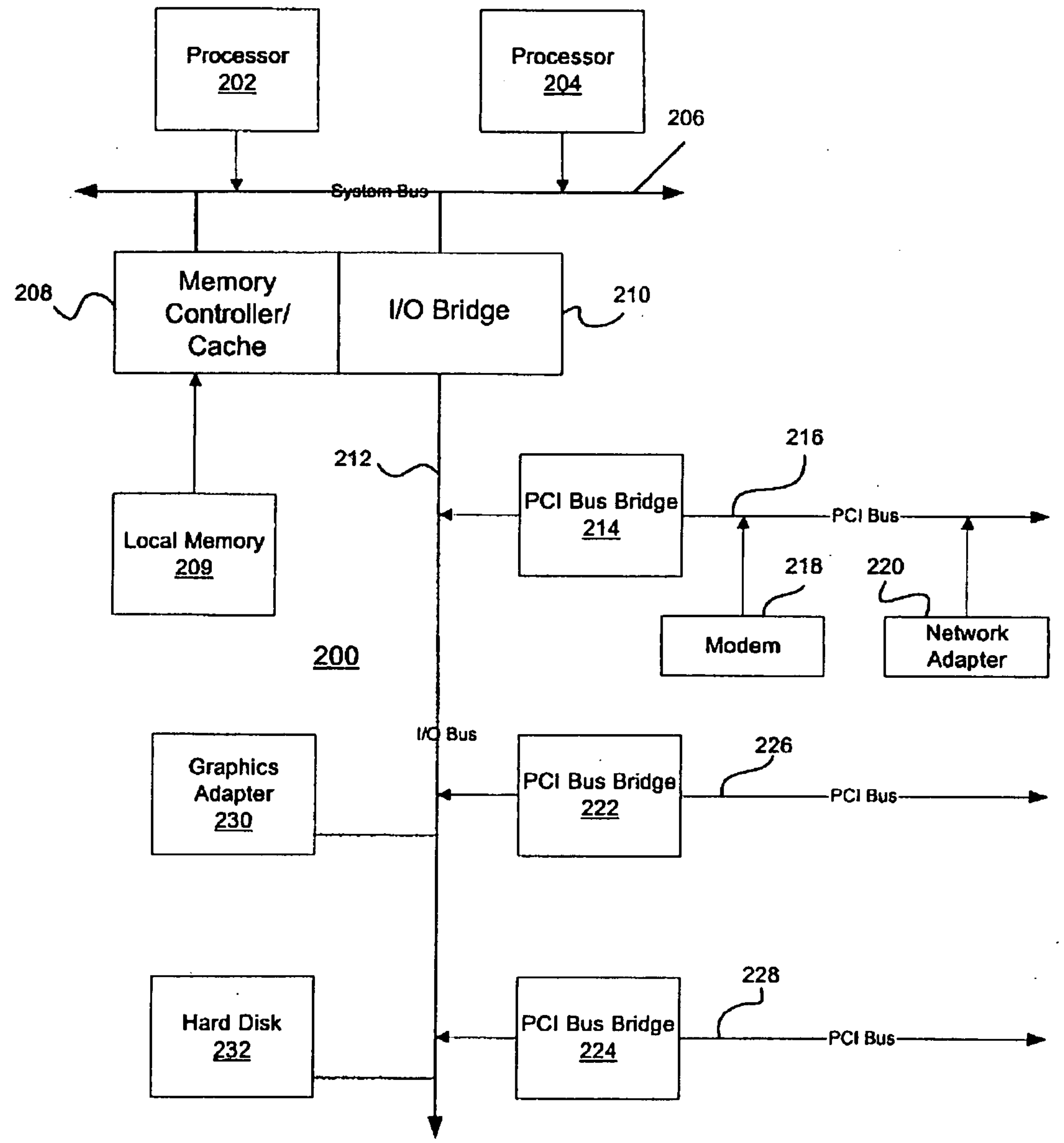
Figure 1



EE318201476US



server

Figure 2

Client

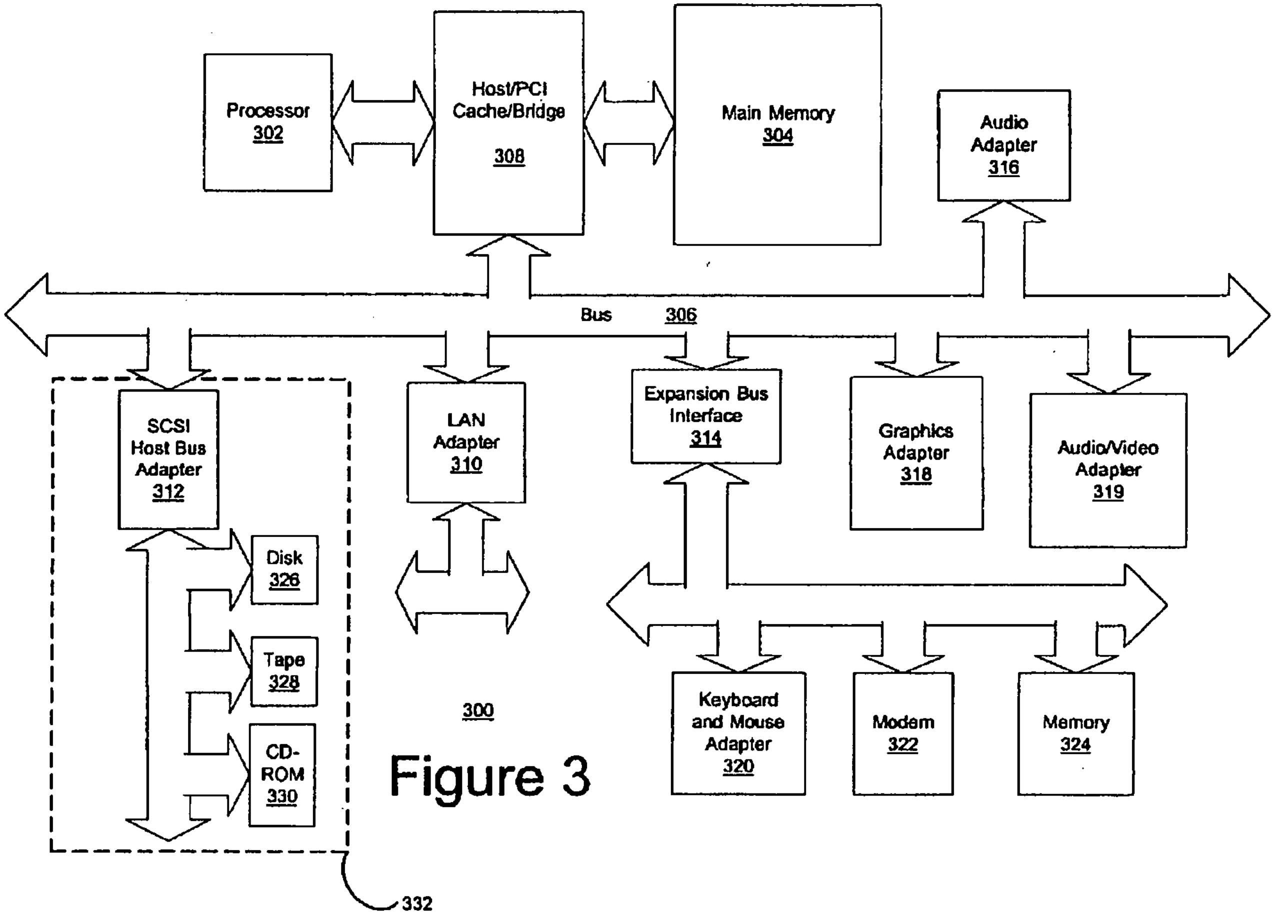


Figure 3

Figure 4

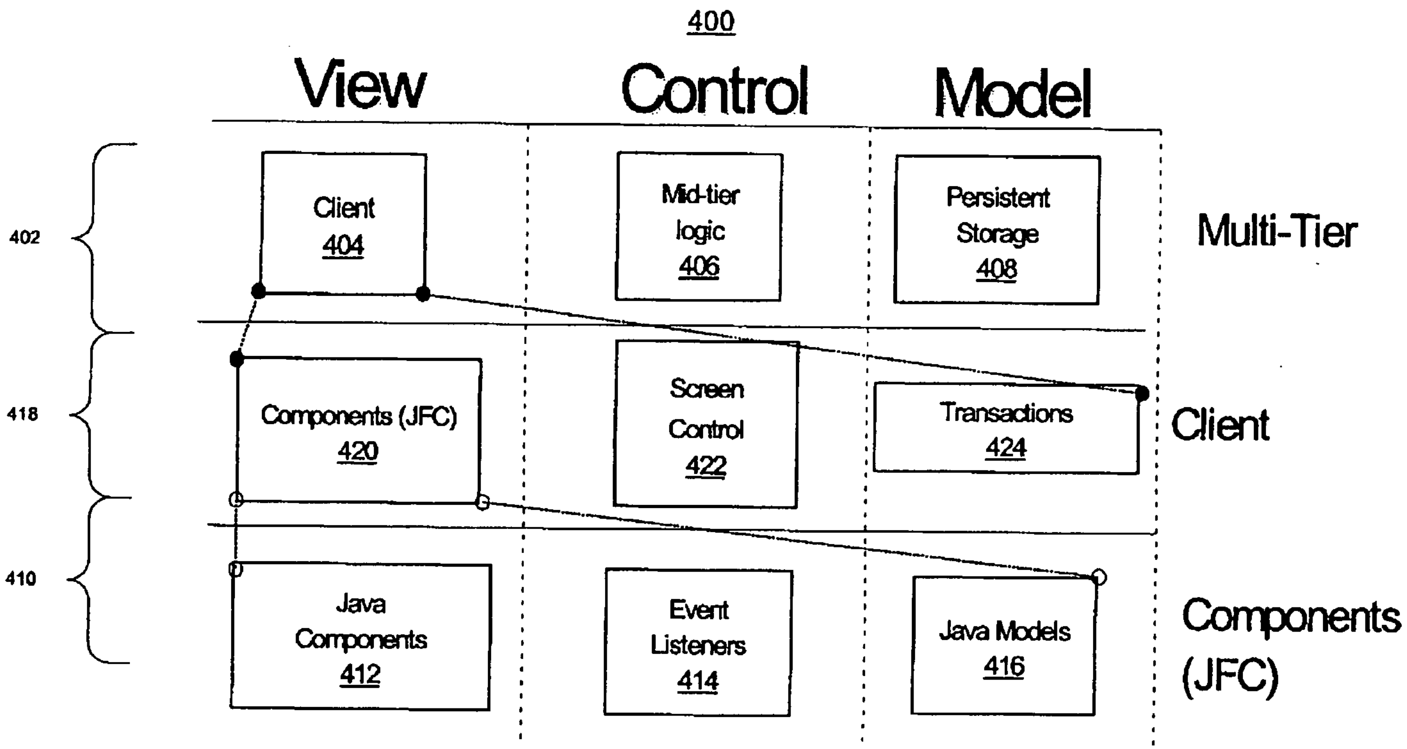
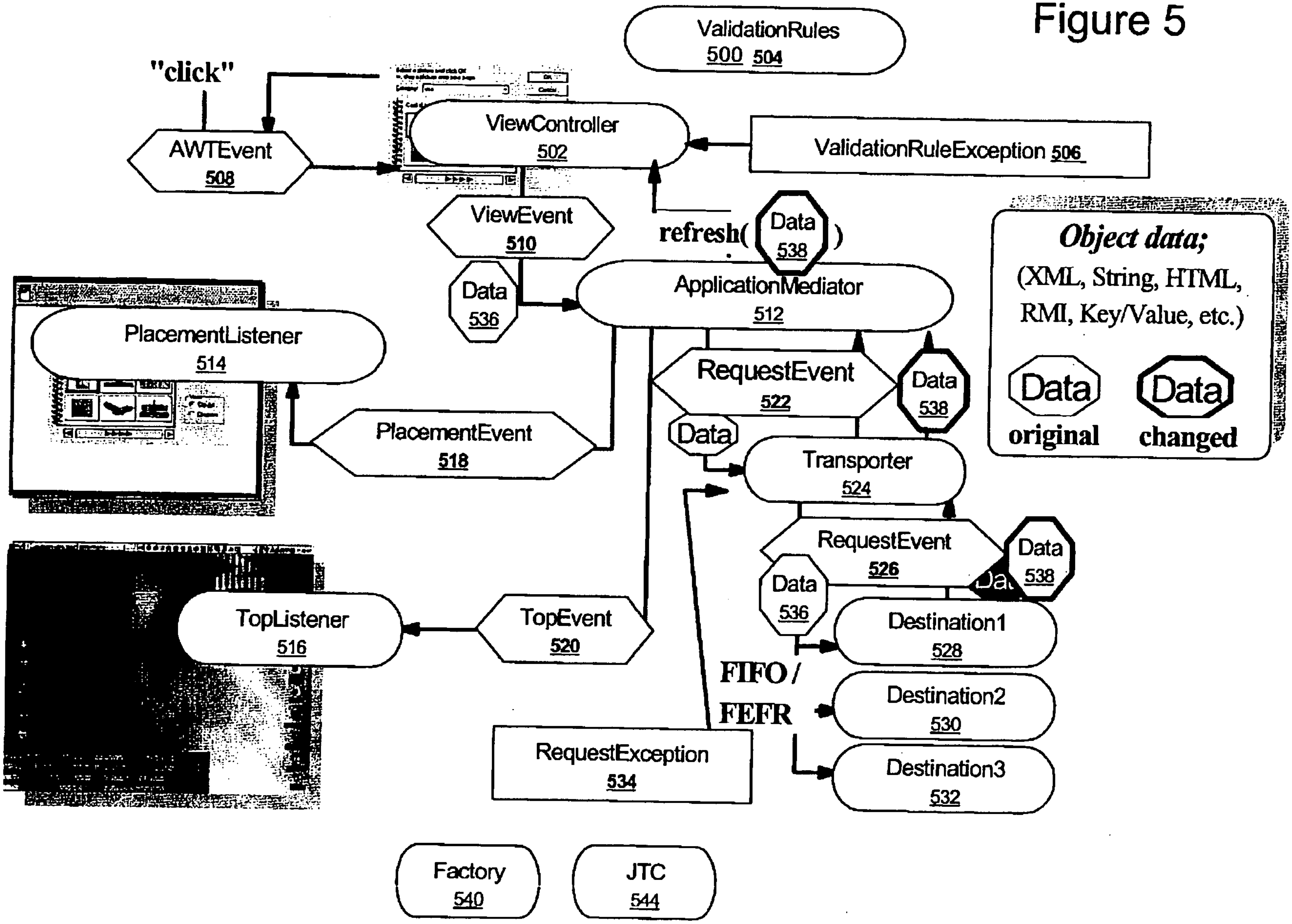

400
View
Control
Model
402
418
410
Client 404
Mid-tier logic 406
Persistent Storage 408
Multi-Tier
Components (JFC) 420
Screen Control 422
Transactions 424
Client
Java Components 412
Event Listeners 414
Java Models 416
Components (JFC)

Figure 5



"click"
AWTEvent
508
ViewController
502
ValidationRules
500 504
ValidationRuleException 506
ViewEvent
510
refresh(Data 538)
Data
536
ApplicationMediator
512
PlacementListener
514
PlacementEvent
518
RequestEvent
522
Data
538
Data
Transporter
524
RequestEvent
526
Data
538
Data
536
Destination1
528
Destination2
530
Destination3
532
FIFO /
FEFR
TopListener
516
TopEvent
520
RequestException
534
Factory
540
JTC
544
Object data;
(XML, String, HTML,
RMI, Key/Value, etc.)
Data
original
Data
changed

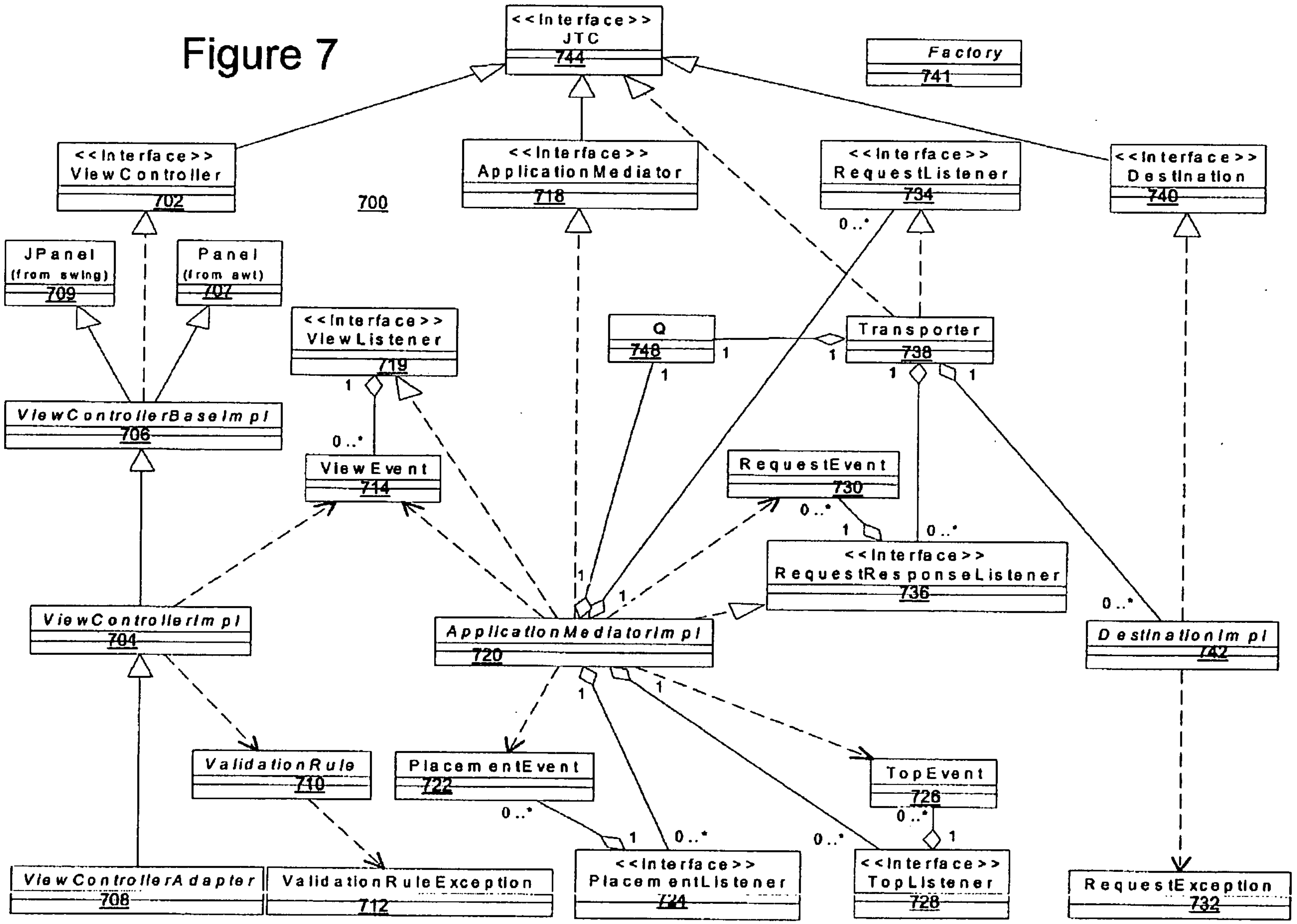
Figure 6

600

Class Hierarchy

```
class java.lang.Object
    interface com.ibm.jtc.ApplicationMediator (extends com.ibm.jtc.JTC)
    class com.ibm.jtc.ApplicationMediatorImpl (implements com.ibm.jtc.ApplicationMediator,
    com.ibm.jtc.ViewListener, com.ibm.jtc.RequestResponseListener)
    interface com.ibm.jtc.Destination (extends com.ibm.jtc.JTC)
    class com.ibm.jtc.DestinationImpl (implements com.ibm.jtc.Destination)
    class java.util.EventObject (implements java.io.Serializable)
      class com.ibm.jtc.PlacementEvent (implements java.io.Serializable)
      class com.ibm.jtc.RequestEvent (implements java.io.Serializable)
      class com.ibm.jtc.TopEvent (implements java.io.Serializable)
      class com.ibm.jtc.ViewEvent (implements java.io.Serializable)
    class com.ibm.jtc.Factory (implements java.io.Serializable)
    interface com.ibm.jtc.JTC (extends java.io.Serializable)
    interface com.ibm.jtc.PlacementListener
    interface com.ibm.jtc.RequestListener
    interface com.ibm.jtc.RequestResponseListener
    class java.lang.Throwable (implements java.io.Serializable)
      class java.lang.Exception
        class com.ibm.jtc.RequestException (implements java.io.Serializable)
        class com.ibm.jtc.ValidationRuleException (implements java.io.Serializable)
    interface com.ibm.jtc.TopListener
    class com.ibm.jtc.Transporter (implements com.ibm.jtc.RequestListener, com.ibm.jtc.JTC)
    class com.ibm.jtc.ValidationRule (implements java.io.Serializable)
    interface com.ibm.jtc.ViewController (extends com.ibm.jtc.JTC)
    interface com.ibm.jtc.ViewListener
```

Figure 7


700
<<Interface>>
JTC
744
Factory
741
<<Interface>>
ViewController
702
<<Interface>>
ApplicationMediator
718
<<Interface>>
RequestListener
734
<<Interface>>
Destination
740
JPanel
(from swing)
709
Panel
(from awt)
707
<<Interface>>
ViewListener
719
Q
748
Transporter
738
ViewControllerBaseImpl
706
ViewEvent
714
RequestEvent
730
<<Interface>>
RequestResponseListener
736
ViewControllerImpl
704
ApplicationMediatorImpl
720
DestinationImpl
742
ValidationRule
710
PlacementEvent
722
TopEvent
726
ViewControllerAdapter
708
ValidationRuleException
712
<<Interface>>
PlacementListener
724
<<Interface>>
TopListener
728
RequestException
732
0..*
1

ViewController

FIGURE 8A

800

Variables

Name	Declaration	Description
copyright	public static final String _copyright	(c) International Business Machines, Inc., 1997 1998 1999. All rights reserved.

FIGURE 8B

802

Methods

Name	Declaration	Description
addViewListener	public abstract void addViewListener (ViewListener listener)	Invoked when a ViewListener is added.
getComponent	public abstract Component get Component ()	Invoked when the ViewController as a component is needed.
getPermissions	public abstract String[] getPermissions ()	Invoked when the ViewController permission keys are needed.
isValid	public abstract boolean isValid ()	Invoked when a ViewController's GUI state needs to be checked to see if it is valid.
isVisible	public abstract boolean isVisible ()	Invoked to see if the ViewController is visible.
refresh	public abstract void refresh (Object data)	Invoked to supply new or changed data.
removeViewListener	public abstract void removeViewListener (ViewListener listener)	Invoked to remove a ViewListener.
setPermissions	public abstract void setPermissions (Hashtable permissions)	Invoked to set the permissions keys and values.
set Properties	public abstract void setProperties (Properties properties)	Invoked to set the properties
setResources	public abstract void setResources (ResourceBundle bundle)	Invoked to set the resources.
setValidationLevel	public abstract void setValidationLevel (int level)	Invoked to give a *hint* to the ViewController as to what validation level to use. The value for level defined in this interface include: NONE = try to do no validation EVENT = try to do validation every event (key) FOCUS = try to do validation on focus change VIEWEVENT = try to do validation before a ViewEvent is issued.
setVisible	public abstract void setVisible (boolean visible)	Invoked to set the visibility.

Figure 9A

ViewControllerImpl

Variables 900

Name	**Declaration**	**Description**
copyright	public static final String copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.
validationLevel	protected int validationLevel	The current validation level
viewEvent	protected ViewEvent viewEvent	A reference to a ViewEvent. Create one ViewEvent reuse it between events.
data	protected Object data	An reference to the data.

Figure 9B

Constructors 902

Name	**Declaration**	**Description**
ViewControllerImpl	public ViewControllerImpl()	Default constructor.

Figure 9C

904

ViewControllerImpl

Methods

Name	Declaration	Description
addViewListener	public final void addViewListener(ViewListener listener)	Add a ViewListener
clear	public void clear()	Clear local state by setting the data reference to null and by removing all ViewListeners.
exit	public void exit()	Get read to exit. Clear local state by setting the data reference to null, removing all ViewListeners and setting view listeners to null.
fireViewEvent	public final void fireViewEvent(ViewEvent event)	If the ViewEvent is not null then send it to all ViewListeners
getComponent	public Component getComponent()	Return the Component that is "this" ViewController. By default, "this" is returned. Redefine this method in ViewControllerBaseImpl when you have a non-java.awt.Component superclass.
getJTCs	public Vector getJTCs()	Return all JTC type objects defined. By default null is returned. Typically, ViewControllers will not return anything.
getPermissions	public String[] getPermissions()	Return a set "keys" that can a management system can use when assigning JTC function based on roles (i.e. group, user). For example, consider the common case of operator override. In a grocery store, if a cashier makes a mistake, a manager inserts a key or enters a password to enable more function on the cash register. The software analogy is that a button may become active or disabled. Suppose the ViewController implements a button labeled "Override" and it is the only component whose state can be visibly altered outside the ViewController. The ViewController writer will return: "Override" In this case, the only options are ENABLE or DISABLE. Suppose these constants are define to be 0x001 and 0x002, respectively. A management system that maintains user priviledges is queried at runtime. The ViewController is then called with setPermissions(keys, values) where keys is "Override" and values is "0x001". The ViewController writer now responds to this request by turning off the button. Instead of hard coding the possible roles, the ViewController simply reacts to key/value settings. By default, nothing is returned.
init	public void init()	Initialize, by default do nothing.
isEnabled	public boolean isEnabled()	Is this ViewController enabled?
isValid	public boolean isValid()	Is the ViewController in a consistent state? This usually means: Do all fields pass ValidationRules? The meaning could also be application specific. This value can can provide other components with the ability to show a visual indicator, such as an X or a check in a tree menu idicating incomplete or partial data. The default value is true.
isVisible	public boolean isVisible()	Is this ViewController visible?
refresh	public void refresh(Object data)	Data objects are being passed in. By default, keep a reference to them. Interpretaion of the data is performed in the subclass. For example, suppose the data being passed is a Customer object. Then a sublcass can perform the following: This can be extended to more complex data types and data type composities (i.e. arrays, Vectors, etc.).
removeViewListener	public final void removeViewListener(ViewListener listener)	Remove a ViewListener
setEnabled	public void setEnabled(boolean toggle)	Enable or disable the ViewController. Remember the state and ask the ViewControllerBaseImpl to handle it.

setPermissions	public void setPermissions(Hashtable permissions)	Given a set of keys and values, update the internal state of the ViewController. The keys and values are supplied via a management system and relate to rules (i.e. users and groups). The possible values in the key/value pairs are application and ViewController specific. For example, create an interface to define the keys and values:
		public interface Customer {
		public static final String DETAILS = "DETAILS";
		public static final String ON = "1";
		public static final String OFF = "0";
		}
		then set the ViewController
		Hashtable permissions = new Hashtable();
		permissions.put(Customer.DETAILS, Customer.ON);
		vc.setPermissions(Permissions);
		The ViewController will interpret the meaning of ON and perform the necessary action, such as active a button. The meaning of keys, values and actions should be defined in a GUI spec. By default, nothing happens.

setProperties	public void setProperties(Properties properties)	Set the properties. Default is to do nothing.
setResources	public void setResources(ResourceBundle bundle)	Set the ResourceBundles. Default is to do nothing.
setValidationLevel	public void setValidationLevel(int level)	Set the validation level to indicate when ValidationRules should be applied Four constants are defined in the ValidationRule class: NONE COMPONENT FOCUS VIEWEVENT This value will be store for the subclass to reference and act. The default value is ValidationRule.NONE.
setVisible	public void setVisible(boolean visible)	Set the ViewController's visiblity on or off. Remember the state and ask the ViewControllerBaseImpl to handle it.
toString	public String toString()	Return the instance class name

Figure 10A

ViewControllerBaseImpl

Variables

Name	Declaration	Description
copyright	public static final String copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.

1000

Figure 10B

Constructors

Name	Declaration	Description
ViewControllerBaseImpl	public ViewControllerBaseImpl()	Default constructor.

1002

Figure 10C

Methods

Name	Declaration	Description
getComponent	public Component getComponent()	By default, return *this*. This works when the superclass is derived from java.awt.Component. Otherwise, override this method and return your own *this*, but be sure to override setEnabled and setVisible also.
setEnabled	public void setEnabled(boolean toggle	By default, passes the call to the super class.
setVisible	public void setVisible(boolean visible)	By default, passed the call to the super class.

Figure 11A

ViewControllerAdapter

1100

Variables

Name	Declaration	Description
_copyright	public static final String _copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.

Figure 11B

1102

Constructors

Name	Declaration	Description
ViewControllerAdapter	public ViewControllerAdapter()	Constructor.

Figure 11C

1104

Methods

Name	Declaration	Description
actionPerformed	public void actionPerformed(ActionEvent e)	Do nothing.
adjustmentValueChanged	public void adjustmentValueChanged(AdjustmentEvent e)	Do nothing.
componentAdded	public void componentAdded(ContainerEvent e)	Do nothing.
componentHidden	public void componentHidden(ComponentEvent e)	Do nothing.
componentMoved	public void componentMoved(ComponentEvent e)	Do nothing.
componentRemoved	public void componentRemoved(ContainerEvent e)	Do nothing.
componentResized	public void componentResized(ComponentEvent e)	Do nothing.
componentShown	public void componentShown(ComponentEvent e)	Do nothing.
focusGained	public void focusGained(FocusEvent e)	Do nothing.
focusLost	public void focusLost(FocusEvent e)	Do nothing.
itemStateChanged	public void itemStateChanged(ItemEvent e)	Do nothing.
keyPressed	public void keyPressed(KeyEvent e)	Do nothing.
keyReleased	public void keyReleased(KeyEvent e)	Do nothing.
keyTyped	public void keyTyped(KeyEvent e)	Do nothing.
mouseClicked	public void mouseClicked(MouseEvent e)	Do nothing.
mouseDragged	public void mouseDragged(MouseEvent e)	Do nothing.
mouseEntered	public void mouseEntered(MouseEvent e)	Do nothing
mouseExited	public void mouseExited(MouseEvent e)	Do nothing
mouseMoved	public void mouseMoved(MouseEvent e)	Do nothing.
mousePressed	public void mousePressed(MouseEvent e)	Do nothing.
mouseReleased	public void mouseReleased(MouseEvent e)	Do nothing.
textValueChanged	public void textValueChanged(TextEvent e)	Do nothing.

Figure 12A

ValidationRule

1200

Variables

Name	Declaration	Description
copyright	public static final String copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.
NONE	public static final int NONE	
COMPONENT	public static final int COMPONENT	
FOCUS	public static final int FOCUS	
VIEWEVENT	public static final int VIEWEVENT	

Figure 12B

1202

Constructors

Name	Declaration	Description
ValidationRule	public ValidationRule()	

Figure 12C

1204

Methods

Name	Declaration	Description
applyEdits	public static String applyEdits(String classNames, String input) throws ValidationRuleException	Given a list of class names, apply each validation rule of the classes to input string and return the formatted result. Parameters: classNames – a comma-separated fully qualified list of concrete AbstractRule classes. input – the input string to apply edit rules to. Returns: the viewable formatted string. Throws: ValidationRuleException if there was an error in applying the edits.
applyNormalize	public static String applyNormalize(String classNames, String input) throws ValidationRuleException	Given a list of class names, apply each normalize rule of the classes to input string and return the transmittable result. Parameters: classNames – a comma-separated fully qualified list of concrete AbstractRule classes. input – the input string to apply normalize rules to. Returns: the transmittable string. Throws: ValidationRuleException
edit	public abstract String edit(String input) throws ValidationRuleException	Subclasses must implement this method to take an input string and apply some edit rule which returns a properly formatted string that can be used to display to the user. Parameters: input-the input string. Returns: the viewable formatted string. Throws: ValidationRuleException if unable to properly format input string.
normalize	public abstract String normalize(String input) throws ValidationRuleException	Subclasses must implement this method to take an input string and apply some normalize rule which returns a properly formatted string that can be used to send data to some server. Parameters: input – the input string. Returns: the transmittable string. Throws: ValidationRuleException if unable to properly format input string.

Figure 12D

1206

```
/**
 * Given a list of class names, apply each validation rule of the classes
 * to input string and return the formatted result.
 *
 * @return the viewable formatted string.
 * @param classNames a comma-separated fully qualified list of concrete AbstractRule classes.
 * @param input the input string to apply edit rules to.
 * @exception ValidationRuleException if there was an error in applying the edits.
 */
public static String applyEdits(String classNames, String input) throws ValidationRuleException {
        int commaIndex = -1;
        int curIndex = 0;
        do {
                commaIndex = classNames.indexOf(',', curIndex);
                if (commaIndex == -1) {
                        commaIndex = classNames.length();
                }
                String className = classNames.substring(curIndex, commaIndex).trim();
                try {
                        ValidationRule rule = (ValidationRule) Factory.newInstance(className);
                        input = rule.edit(input);
                } catch (ValidationRuleException re) {
                        throw re;
                } catch (Exception e) {
                        throw new ValidationRuleException("Rule class '" + className + "' not found.");
                }
                curIndex = commaIndex + 1;
        } while (curIndex < classNames.length());
        return input;
}
```

Figure 13A

ValidationRuleException

1300

Variables

Name	Declaration	Description
copyright	public static final String copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.

Figure 13B

1302

Constructors

Name	Declaration	Description
ValidationRuleException	public ValidationRuleException()	Default constructor.
ValidationRuleException	public ValidationRuleException(String s)	Constructor with a message to the rule exception.

Figure 14A

ViewEvent

1400

Variables

Name	Declaration	Description
_copyright	public static final String _copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.
VIEWEVENT_FIRST	public static final int VIEWEVENT_FIRST	
OK	public static final int OK	
DONE	public static final int DONE	
OPEN	public static final int OPEN	
CLOSE	public static final int CLOSE	
CANCEL	public static final int CANCEL	
EXIT	public static final int EXIT	
FILE	public static final int FILE	
SAVE	public static final int SAVE	
SAVEAS	public static final int SAVEAS	
ERROR	public static final int ERROR	
WARNING	public static final int WARNING	
RETURN	public static final int RETURN	
LOAD	public static final int LOAD	
NOTIFY	public static final int NOTIFY	
NOTIFY2	public static final int NOTIFY2	
INFO	public static final int INFO	
SETUP	public static final int SETUP	
PRINT	public static final int PRINT	

Figure 14B

ViewEvent
(continued)

1400

Variables

Name	Declaration	Description
TITLEMESSAGE	public static final int TITLEMESSAGE	
STATUSMESSAGE	public static final int STATUSMESSAGE	
ERRORMESSAGE	public static final int ERRORMESSAGE	
SUGGESTIONMESSAGE	public static final int SUGGESTIONMESSAGE	
NEXT	public static final int NEXT	
PREVIOUS	public static final int PREVIOUS	
FIRST	public static final int FIRST	
LAST	public static final int LAST	
START	public static final int START	
BEGIN	public static final int BEGIN	
END	public static final int END	
PAUSE	public static final int PAUSE	
STOP	public static final int STOP	
RESTART	public static final int RESTART	
SUBMIT	public static final int SUBMIT	
BACKSPACE	public static final int BACKSPACE	
INSERT	public static final int INSERT	

Figure 14C

ViewEvent
(continued)

1400

Variables

Name	Declaration	Description
HOME	public static final int HOME	
PGUP	public static final int PGUP	
PGDN	public static final int PGDN	
LEFT	public static final int LEFT	
RIGHT	public static final int RIGHT	
UP	public static final int UP	
DOWN	public static final int DOWN	
LIST	public static final int LIST	
MORE	public static final int MORE	
ADD	public static final int ADD	
DELETE	public static final int DELETE	
MODIFY	public static final int MODIFY	
NEW	public static final int NEW	
EDIT	public static final int EDIT	
COPY	public static final int COPY	
CUT	public static final int CUT	
PASTE	public static final int PASTE	
UNDO	public static final int UNDO	
REMOVE	public static final int REMOVE	
PLUS	public static final int PLUS	
MINUS	public static final int MINUS	
INCREMENT	public static final int INCREMENT	
DECREMENT	public static final int DECREMENT	
CHANGED	public static final int CHANGED	

Figure 14D

ViewEvent
(continued)

1400

Variables

Name	Declaration	Description
FILL	public static final int FILL	
EMPTY	public static final int EMPTY	
READY	public static final int READY	
VIEW	public static final int VIEW	
DETAILS	public static final int DETAILS	
READ	public static final int READ	
WRITE	public static final int WRITE	
SEARCH	public static final int SEARCH	
FIND	public static final int FIND	
HELP	public static final int HELP	
HINT	public static final int HINT	
TRAIN	public static final int TRAIN	
TEACH	public static final int TEACH	
SUGGEST	public static final int SUGGEST	
VIEWEVENTTEST1	public static final int VIEWEVENTTEST1	
VIEWEVENTTEST2	public static final int VIEWEVENTTEST2	
VIEWEVENTTEST3	public static final int VIEWEVENTTEST3	
VIEWEVENT LAST	public static final int VIEWEVENT LAST	
consumed	protected boolean consumed	Is event still valid
timestamp	protected long timestamp	Time stamp when event is fired.
data	protected Object data	Data reference

Figure 14E

1402

Constructors

Name	Declaration	Description
ViewEvent	public ViewEvent()	Constructs a ViewEvent
ViewEvent	public ViewEvent(Object source)	Constructs a ViewEvent
ViewEvent	public ViewEvent(Object source, int major)	Constructs a ViewEvent object with the specified source object and code;
ViewEvent	public ViewEvent(Object source, int major, int minor, Object data)	Constructs a ViewEvent object with the specified source object and code;
ViewEvent	public ViewEvent(Object source, int major, Object data)	Constructs a ViewEvent object with the specified source object and code;

Figure 14F

ViewEvent
(continued)

1404

Methods

Name	Declaration	Description
consume	public final void consume()	Consume this event.
getData	public Object getData()	Return the data
getMajor	public final int getMajor()	Return the major event code
getMinor	public final int getMinor()	Return the event option
getSource	public final Object getSource()	Gets the event source Overrides: getSource in class EventObject
getTimestamp	public long getTimestamp()	Get the timestamp when the event was fired. By default, this was set by JTC.
isConsumed	public final boolean isConsumed()	Is the event consumed?
setConsumed	public final void setConsumed(boolean consumed)	Turn event consumed on/off.
setData	public void setData(Object data)	Sets the data
setMajor	public final void setMajor(int code)	Sets the event code
setMinor	public final void setMinor(int code)	Sets the event option
setSource	public final void setSource(Object source)	Sets the event source
setTimestamp	public void setTimestamp(long time)	Set the timestamp when the event is fired. By default, this is set by JTC.
toString	public String toString()	Returns a string representation of the object. The class of the event and the reason for the event is returned.

Figure 15A

ViewListener

1500

Variables

Name	Declaration	Description
_copyright	public static final String _copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.

Figure 15B

1502

Methods

Name	Declaration	Description
viewEventPerformed	public abstract void viewEventPerformed(ViewEvent event)	Invoked when a ViewEvent has been fired.

ApplicationMediator

Variables

1600

FIGURE 16A

Name	Declaration	Description
copyright	public static final String _copyright	(c) International Business Machines, Inc., 1997 1998 1999. All rights reserved.

Methods

1602

FIGURE 16B

Name	Declaration	Description
addPlacementListener	public abstract void addPlacementListener (PlacementListener listener)	Invoked when a PlacementListener is added.
addRequestListener	public abstract void addRequestListener (RequestListener listener)	
addTopListener	public final void addTopListener (TopListener listener)	Add a TopListener.
addViewListener	public abstract void addViewListener (ViewListener listener)	Invoked when a ViewListener is added.
getPermissions	public abstract String[] getPermissions()	Invoked when the ApplicationMediator permission keys are needed.
init	public abstract void init (ApplicationMediator applicationMediator)	Invoked when an ApplicationMediator should be initialized based on another ApplicationMediator's contents.
isValid	public abstract boolean isValid()	Invoked when the ApplicationMediator's state needs to be checked to see if it is valid.
is Visible	public abstract boolean isVisible()	Invoked to see if the ApplicationMediator is visible.
refresh	public abstract void refresh (Object data)	Invoked to supply new or changed data.
removePlacementListener	public abstract void removePlacementListener (PlacementListener listener)	Invoked to remove a PlacementListener.
removeRequestListener	public abstract void removeRequestListener (RequestListener listener)	Invoked to remove a RequestListener
removeTopListener	public final void removeTopListener (TopListener listener)	Removes the TopListener.
remove ViewListener	public abstract void removeViewListener (ViewListener listener)	Invoked to remove a ViewListener.
setPermissions	public abstract void setPermissions (Hashtable permissions)	Invoked to set the permissions keys and values.
setProperties	public abstract void setProperties (Properties properties)	Invoked to set the properties.
setResources	public abstract void setResources (ResourceBundle bundle)	Invoked to set the resources.
setVisible	public abstract void setVisible (boolean visible)	Invoked to set the visibility.

Figure 17A

ApplicationMediatorImpl

1700

Variables

Name	Declaration	Description
placementListeners	protected Vector placementListeners	The PlacementListeners.
topListeners	protected TopListener topListener	The TopListener
requestListeners	protected Vector requestListeners	The RequestListeners.
viewListeners	protected Vector viewListeners	The ViewEventListeners
viewControllers	protected Vector viewControllers	Whenever view controllers are created, it is by convention they will be added to this array.
applicationMediators	protected Vector applicationMediators	Whenever application mediators are created, it is by convention they will be added to this array.
data	protected Object data	This is a reference to the system data model.
requestEvent	protected RequestEvent requestEvent	This is a reference to a RequestEvent.

Figure 17B

1702

Constructors

Name	Declaration	Description
ApplicationMediatorImpl	public ApplicationMediatorImpl()	Constructor. By changing commented code, you can switch between threading styles 1 and 2.

Figure 17C

ApplicationMediatorImpl

1704

Methods

Name	Declaration	Description
addPlacementListener	public final void addPlacementListener(PlacementListener listener)	Add a PlacementListener.
addRequestListener	public final void addRequestListener(RequestListener listener)	Add a RequestListener.
addTopListener	public final void add TopListener (TopListener listener)	Add a TopListener.
addViewListener	public final void addViewListener(ViewListener listener)	Add a ViewListener.
clear	public void clear()	Clear the ApplicationMediator by clearing all allocated ViewControllers and ApplicationMediators. All data is set to null, but lists are not destroyed. A 'cleared' ApplicationMediator can be used again. If this method is overriden in a subclass, be sure to invoke super.clear();
exit	public void exit()	Exit the ApplicationMediator by exiting all allocated ViewControllers and ApplicationMediators. All data is set to null, and lists are destroyed. An 'exited' ApplicationMediator cannot be used again. If this method is overriden in a subclass, be sure to invoke super.exit();
firePlacementEvent	protected final void firePlacementEvent(PlacementEvent event)	Notify the PlacementListeners.
fireRequestEvent	protected final void fireRequestEvent(RequestEvent event) throws RequestException	Notify the RequestListeners -synchronous
fireRequestEvent	protected final void fireRequestEvent(RequestEvent event, RequestResponseListener caller) throws RequestException	Notify the RequestListeners - asynchronous.
fireTopEvent	protected final void fireTopEvent (TopEvent event)	Notify the TopListeners.
fireTopListener		
fireViewEvent	protected final void fireViewEvent(ViewEvent event)	Notify the ViewtListeners.
getAM	protected ApplicationMediator getAM(int i)	Return the i'th ApplicationMediator
getJTCs	public Vector getJTCs()	Return a vector of all ThinClient objects. By default, this is a Vector containing the created ViewControllers and ApplicationMediators.
getPermissions	public String[] getPermissions()	Get the settable permission keys. By default, return the class names of all allocated ViewControllers and ApplicationMediators.
getVC	protected ViewController getVC(int i)	Return the i'th ViewController
init	public void init()	Initialize the ApplicationMediator, nothing to do by default.
init	public void init(ApplicationMediator applicationMediator)	Initialize the ApplicationMediator using the listeners of an existing ApplicationMediator.
initApplicationMediators	public final void initApplicationMediators(String classnames[]) throws ClassNotFoundException, InstantiationException, IllegalAccessException	For each ApplicationMediator classname, load it, new it and add myself as a ViewEvent. The Factory class is used as a helper class.
initViewControllers	public final void initViewControllers(String classnames[]) throws ClassNotFoundException, InstantiationException, IllegalAccessException	For each ViewController classname, load it, new it and add myself as a ViewEvent. The Factory class is used as a helper class.
isEnabled	public boolean isEnabled()	Is the ApplicationController enabled?
isValid	public boolean isValid()	Return the AND'ed value of calling isValid on ApplicationMediators and ViewControllers.
isVisible	public boolean isVisible()	Is the ApplicationController visible? Hardly, since it is a non visible class. But this looks to see if any of its ViewControllers are visible. Not really, they were all set to visible/invisible via the setVisible method and we remembered the state to return here.
processViewEvent	public abstract void processViewEvent(ViewEvent e)	Deliver the ViewEvent to the subclass via this method.
refresh	public void refresh(Object data)	When new data arrives allow the ViewControllers and ApplicationControllers to be refreshed also.

Figure 17D

ApplicationMediatorImpl
(continued)

1704

Methods

Name	Declaration	Description
removePlacementListener	public final void removePlacementListener(PlacementListener listener)	Removes the PlacementListener.
removeRequestListener	public final void removeRequestListener(RequestListener listener)	Removes the RequestListener.
removeViewListener	public final void removeViewListener(ViewListener listener)	Removes the ViewListener.
requestException	public void requestException(RequestException yikes)	Called back because an asynchronous request has thrown an Exception. By default, print the message to System.err.
requestResponse	public void requestResponse(RequestEvent respons	Called back with the results of an asynchronous request. By default, call refresh with the data in the respo
run2	public final void run2()	This method is used in style 1 threading. Rename this to run() and uncomment the code as described in the class javadoc.
setAM	public void setAM(ApplicationMediator applicationMediator, int i)	Set the i'th ApplicationMediator
setEnabled	public void setEnabled(boolean toggle)	Call setEnabled on each ViewController and ApplicationMediator.
setPermissions	public void setPermissions(Hashtable permissions)	Set the permissions. By default, call setPermissions on each ViewController and ApplicationMediator
setProperties	public void setProperties(Properties propertie	Set the properties.. By default, call setProperties on each ViewController and ApplicationMediator
setResources	public void setResources(ResourceBundle bundle)	Set the resources.. By default, call setResources on each ViewController and ApplicationMediator
setVC	public void setVC(ViewController viewController, int i)	Set the i'th ViewController
setVisible	public void setVisible(boolean visible)	Set visible on each ViewController and ApplicationMediator
toString	public String toString()	Return the Class name of the ApplicationController instance
viewEventPerformed	public void viewEventPerformed(ViewEvent e)	A ViewEvent is delivered. Process it using Threading style 1 or 2. In the end, the processViewEvent will be called on the subclass.

ApplicationMediatorImpl.exit(): AUS8-1999-0694

```
/**
 * Exit the ApplicationMediator by exiting all allocated ViewControllers
 * and ApplicationMediators.  All data is set to null, and lists are
 * destroyed.  An 'exited' ApplicationMediator cannot be used again.
 * If this method is overriden in a subclass, be sure to invoke
 * super.exit();
 **/
public void exit() {
      synchronized (this) {
            /* Used for style 1 event dispatching. Leave this code commented. */
            //if (this.eventThread != null) {
            //     try {
            //           eventThread.stop();
            //     } catch (Exception e) {
            //     }
            //}


            /* Used for style 2 event dispatching. Leave this code commented. */
            for (int i = 0; i < runningThreads.size(); i++) {
                  ((ApplicationMediatorThread) runningThreads.elementAt(i)).stop();
            }
            runningThreads.removeAllElements();
            viewListeners.removeAllElements();
            try {
                  for (int i = 0; i < viewControllers.size(); i++) {
                        ((ViewController) viewControllers.elementAt(i)).setEnabled(false);
                        ((ViewController) viewControllers.elementAt(i)).exit();
                  }
                  for (int i = 0; i < applicationMediators.size(); i++) {
                        ((ApplicationMediator) applicationMediators.elementAt(i)).setEnabled(false);
                        ((ApplicationMediator) applicationMediators.elementAt(i)).exit();
                  }
            } catch (Exception noProblem) {
            }
            viewControllers = null;
            applicationMediators = null;
            runningThreads = null;
            runningThreads = null;
            data = null;
      }
}
```

1706

Figure 17E

ApplicationMediatorImpl.clear():AUS8-1999-0694

```
/**
 * Clear the ApplicationMediator by clearing all allocated ViewControllers
 * and ApplicationMediators.  All data is set to null, but lists are
 * not destroyed.  A 'cleared' ApplicationMediator can be used again.
 * If this method is overriden in a subclass, be sure to invoke
 * super.clear();
 */
public void clear() {
      synchronized (this) {

            /* Used for style 1 event dispatching. Leave this code commented. */
            //if (this.eventThread != null) {
            //      try {
            //            eventThread.stop();
            //      } catch (Exception e) {
            //      }
            //}

            /* Used for style 2 event dispatching. Leave this code commented. */
            for (int i = 0; i < runningThreads.size(); i++) {
                  ((ApplicationMediatorThread) runningThreads.elementAt(i)).stop();
            }
            runningThreads.removeAllElements();

            //
            try {
                  for (int i = 0; i < viewControllers.size(); i++) {
                        ((ViewController) viewControllers.elementAt(i)).setEnabled(false);
                        ((ViewController) viewControllers.elementAt(i)).clear();
                  }
                  for (int i = 0; i < applicationMediators.size(); i++) {
                        ((ApplicationMediator) applicationMediators.elementAt(i)).setEnabled(false);
                        ((ApplicationMediator) applicationMediators.elementAt(i)).clear();
                  }
            } catch (Exception noRealProblemo) {
            }
            viewControllers = null;
            applicationMediators = null;
            data = null;
            viewListeners.removeAllElements();
      }
}
```

1708

Figure 17F

```
/**
 * Initialize the ApplicationMediator using the listeners of an
 * existing ApplicationMediator.
 */
public void init(ApplicationMediator applicationMediator) {
        if (applicationMediator instanceof ApplicationMediatorImpl) {
                ApplicationMediatorImpl a = (ApplicationMediatorImpl) applicationMediator;
                requestListeners = (Vector) a.requestListeners.clone();
                placementListeners = (Vector) a.placementListeners.clone();
                topListeners = (Vector) a.topListeners.clone();
                addViewListener(a);
        }
        init();
}
```

1710

Figure 17G

Figure 17H

1712

```
/**
 * When new data arrives allow the ViewControllers
 * and ApplicationControllers to be refreshed also.
 */
public void refresh(Object data) {
        this.data = data;
        try {
                synchronized (viewControllers) {
                        for (int j = 0; j < viewControllers.size(); j++) {
                                ((ViewController) viewControllers.elementAt(j)).
                                        refresh(data);
                        }
                }
        } catch (Exception noRealProblem) {
        }
        try {
                synchronized (applicationMediators) {
                        for (int j = 0; j < applicationMediators.size(); j++) {
                                ((ApplicationMediator) applicationMediators.
                                        elementAt(j)).refresh(data);
                        }
                }
        } catch (Exception noRealProblem) {
        }
}
```

```
/**
 * A ViewEvent is delivered.  Process it using Threading style 1 or 2.  In
 * the end, the processViewEvent will be called on the subclass.
 */
public void viewEventPerformed(ViewEvent e) {
    /* Used for style 2 event dispatching, start an inner classs thread */
    ApplicationMediatorThread t = new ApplicationMediatorThread(e);
    runningThreads.addElement(t);
    t.start();

    /* Used for style 1 event dispatching. Leave this code commented.  */
    //ViewEvent saved = saveViewEvent(e);
    //if (eventThread == null || !eventThread.isAlive()) {
    //   finished = false;
    //   eventThread = new Thread(this);
    //   eventThread.start();                        1714
    //}
    //synchronized (this) {
    //   notify();
    //}
}
```

Figure 17I

```
/**
 * This method is used in style 1 threading.  Rename this to run()
 * and uncomment the code as described in the class javadoc.
 */
public final void run2() {
     /* Used for style 1 event dispatching. Leave this code commented. */
     /*
     while (true) {
               ViewEvent event = null;
          event = getViewEvent();
          if (event != null) {
               handleViewEvent(event);
          } else {
               waitForEvent();
               if (finished) {
               //something went wrong with the thread so hose this loop
               break;
               }
          }
     }
     */
}
```

1714

Figure 17J

```
/**
* Private class to handle executions of ViewEvents() on another thread.
**/
private class ApplicationMediatorThread extends Thread {
     /**
     * The current event
     **/
     private ViewEvent event;
     /**
     * Create an ApplicationMediatorThread to process the ViewEvent
     **/
     public ApplicationMediatorThread(ViewEvent event) {
          super();
          this.event = event;
     }
     /**
     * Just call the handleViewEvent method that the subclass will override
     **/
     public void run() {
          processViewEvent(event);
     }
}
```

1714

Figure 17K

```
/**
 * Save the current ViewEvent on a Q
 */
private final ViewEvent saveViewEvent(ViewEvent e) {
     /* Used for style 1 event dispatching. Leave this code commented. */
     //return viewEventQueue.add(e);
     return null;
}                                                                    1714

/**
 * Method: return the first view event saved. Used by the Q'ing system.
 */
private ViewEvent getViewEvent() {
     /* Used for style 1 event dispatching. Leave this code commented. */
     //return (ViewEvent) viewEventQueue.remove();
     return null;
}
```

Figure 17L

Figure 18A

1800 PlacementEvent

Variables

Name	Declaration	Description
_copyright	public static final String _copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.
PLACEMENTEVENT_FIRST	public static final int PLACEMENTEVENT_FIRST	
ADD	public static final int ADD	
REMOVE	public static final int REMOVE	
MODIFY	public static final int MODIFY	
PLACEMENTEVENT_LAST	public static final int PLACEMENTEVENT_LAST	
major	protected int major	The placementevent code
minor	protected int minor	the placementevent option
component	protected Object component	Component Reference
data	protected Object data	Data reference

Figure 18B

1802

Constructors

Name	Declaration	Description
PlacementEvent	public PlacementEvent()	Constructs a PlacementEvent
PlacementEvent	public PlacementEvent(Object source, Object component)	Constructs a PlacementEvent
PlacementEvent	public PlacementEvent(Object source, Object component, int major)	Constructs a PlacementEvent
PlacementEvent	public PlacementEvent(Object source, Object component, int major, int minor)	Constructs a PlacementEvent
PlacementEvent	public PlacementEvent(Object source, Object component, int major, int minor, Object data)	Constructs a PlacementEvent

Figure 18C

1804

Methods

Name	Declaration	Description
getComponent	public final Component getComponent()	Return the Component
getData	public final Object getData()	Return the data
getMajor	public final int getMajor()	Return the major code
getMinor	public final int getMinor()	Return the minor code
getSource	public final Object getSource()	Gets the event source
setComponent	public final void setComponent(Component component)	Sets the Component
setData	public final void setData(Object data)	Set the data
setMajor	public final void setMajor(int code)	Set the major code
setMinor	public final void setMinor(int code)	Sets the minor code
setSource	public final void setSource(Object source)	Set the event source
toString	public String toString()	Returns a string representation of the object.

PlacementListener

FIGURE 19A

1900

Variables

Name	Declaration	Description
copyright	public static final String copyright	(c) International Business Machines, Inc., 1997 1998 1999. All rights reserved.

FIGURE 19B

1902

Methods

Name	Declaration	Description
placementEveatPerformed	public abstract void placementEventPerformed (PlacementEvent event)	Invoked when we are being called to add/remove/modify a component. Do it.

TopEvent

FIGURE 20A

2000

Variables

Name	Declaration	Description
copyright	public static final String _copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.
TOPEVENT_FIRST	public static final int TOPEVENT_FIRST	
EXIT	public static final int EXIT	
BROWSER	public static final int BROWSER	
TITLE	public static final int TITLE	
STATUS	public static final int STATUS	
OS	public static final int OS	
A	public static final int A	
B	public static final int B	
C	public static final int C	
D	public static final int D	
E	public static final int E	
F	public static final int E	
TRACE	public static final int TRACE	
DEBUG	public static final int DEBUG	
LOG	public static final int LOG	
HOOKAWT	public static final int HOOKAWT	
HOOKJTC	public static final int HOOKJTC	
TOPEVENT_LAST	public static final int TOPEVENT_LAST	
TEAM	public static final int TEAM	
WIN	public static final int WIN	
EXECUTE	public static final int execute	
consumed	protected boolean consumed	Is event still valid?
data	protected Object data	This is a loose reference to the data model. We don't care what the class shape is and we only reference it via the interface that it must implement.

TopEvent

FIGURE 20B

2002

Constructors

Name	Declaration	Description
TopEvent()	public TopEvent ()	Default constructor for a Request.
TopEvent(Object)	public TopEvent (Object source)	Construct with the given source and default major and minor values.
TopEvent(Object, int)	public TopEvent (Object source, int major)	Create a Request with a source, major and minor codes.
TopEvent(Object, int, int)	public TopEvent (Object source, int major, int minor)	Create a Request with major and minor codes.
TopEvent(Object, int, int, Object)	public TopEvent (Object source, int major, int minor, Object data)	Create a Request with a source, major and minor codes, and some data. If source is null, an InvalidArgumentException will be thrown.

FIGURE 20C

2004

Methods

Name	Declaration	Description
consume	public final void consume ()	Consume this event
getData	public final Object get Data ()	Return the reference to the data.
getMajor	public final int getMajor ()	Get the major code.
getMinor	public final int getMinor ()	Get the minor code.
getSource	public final Object getSource ()	Gets the event source. **Overrides:** getSource in class EventObject
isConsumed	public final boolean isConsumed ()	Is the event consumed?
setConsumed	public final void setConsumed (boolean consumed)	Turn event consumed on or off.
setData	public final void setData (Object data)	Set the data.
setMajor	public final void setMajor (int major)	Set the major code.
setMinor	public final void setMinor (int minor)	Set the minor code. This is always a String.
setSource	public final void setSource (Object source)	Sets the event source.
toString	public String toString ()	Show a String representation of the Request in the format of "TopEvent(major,minor)"

TopListener

FIGURE 21A

2100

Variables

Name	Declaration	Description
copyright	public static final String _copyright	(c) International Business Machines, Inc., 1997 1998 1999. All rights reserved.

FIGURE 21B

2102

Methods

Name	Declaration	Description
exec	public abstract void exec (Object programInformation)	Invoked to execute a desktop program. The parameter programInformation can be a complex object with lots of data. For example: String[] params = {"netscape.exe", "http://www.ibm.com"}; aTopListener.exec(params). Another usage might be to interact with JavaScript under a browser. It is up to the TopListener implementer to understand what the params mean. Do not create a language with a language. This method should only be defined to support legacy environments or corporate desktop rules. Consider using a RequestEvent for more complex requirements.
exit	public abstract void exit()	Invoked to exit a JTC application. Never let a program perform its own "exit". This shuts the JVM down. The implementer of TopListener will know the appropriate actions to take to exit from an application on a corporate desktop.
message	public abstract void message (Object messageInfo)	Invoked to show a business specific message. Try to isolate calls to the browser here.
title	public abstract void title (Object titleInfo)	Invoked to display a business specific title. Try to isolate calls to a browser or a desktop program to display titles here.
topEventPerformed	public abstract void topEventPerformed (TopEvent event)	Invoked when we are being called to perform a top desktop function.

RequestEvent

Figure 22A

2200

Variables

Name	Declaration	Description
_copyright	public static final String _copyright	© International Business Machines Inc., 1997 1998 1999. All rights reserved.
consumed	protected boolean consumed	Is event still valid
data	protected Object data	This is a loose reference to the data model. We don't care what the class shape is and we only reference it via the interface that it must implement.

Figure 22B

2202

Constructors

Name	Declaration	Description
RequestEvent	public RequestEvent()	Default constructor for a Request.
RequestEvent	public RequestEvent(Object source)	Construct with the given source and default major and minor values.
RequestEvent	public RequestEvent(Object source, String major)	Create a Request with a source, major and minor codes
RequestEvent	public RequestEvent(Object source, String major, String minor)	Create a Request with major and minor codes
RequestEvent	public RequestEvent(Object source, String majorCode, String minorCode, Object data)	Create a Request with a source, major and minor codes, and some data. If source is null, an InvalidArgumentException will be thrown.

Figure 22C

2204

Methods

Name	Declaration	Description
consume	public final void consume()	Consume this event.
getData	public final Object getData()	Return the reference to the data.
getMajor	public final String getMajor()	Get the major code. This is always a String.
getMinor	public final String getMinor()	Get the minor code. This is always a String.
getSource	public final Object getSource()	Gets the event source
getStatus	public final String getStatus()	Return the status
isConsumed	public final boolean isConsumed()	Is the event consumed?
setConsumed	public final void setConsumed(boolean consumed)	Turn event consumed on or off.
setData	public final void setData(Object data)	Set the data
setMajor	public final void setMajor(String major)	Set the major code. This is always a String.
setMinor	public final void setMinor(String minor)	Set the minor code. This is always a String.
setSource	public final void setSource(Object source)	Sets the event source
setStatus	public final void setStatus(String message)	Append a message to the status
toString	public String toString()	Show a String representation of the Request in the format of "RequestEvent(major,minor)"

Figure 23A

RequestException

2300

Variables

Name	Declaration	Description
copyright	public static final String copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.

Figure 23B

2302

Constructors

Name	Declaration	Description
RequestException	public RequestException()	Default constructor.
RequestException	public RequestException(String s)	Constructor with a message to the request exception.
RequestException	public RequestException(Throwable target)	Constructor with a throwable target
RequestException	public RequestException(Throwable target, String s)	Constructor with a throwable target and a message

Figure 23C

2304

Methods

Name	Declaration	Description
getTargetException	public Throwable getTargetException()	Get the target throwable
setTargetException	public void setTargetException(Throwable target)	Set the target throwable
toString	public String toString()	String version

RequestListener

FIGURE 24A

Variables

2400

Name	Declaration	Description
copyright	public static final String _copyright	(c) International Business Machines, Inc., 1997 1998 1999. All rights reserved.

FIGURE 24B

Methods

2402

Name	Declaration	Description
requestEventPerformed	public abstract void requestEventPerformed (RequestEvent request) throws RequestException	Invoked for a synchronous RequestEvent.
requestEventPerformed	public abstract void requestEventPerformed (RequestEvent request, RequestResponseListener listener) throws RequestException	Invoked for an asynchronous RequestEvent.

RequestResponseListener

FIGURE 25A

2500

Variables

Name	Declaration	Description
copyright	public static final String _copyright	(c) International Business Machines, Inc., 1997 1998 1999. All rights reserved.

FIGURE 25B

2502

Methods

Name	Declaration	Description
requestException	public abstract void requestException (RequestException yikes)	Invoked when an exception occurred during processing of an asynchronous RequestEvent.
requestResponse	public abstract void requestResponse (RequestEvent result)	Invoked when the processing of an asynchronous RequestEvent was successful.

Figure 26A

Transporter

2600

Variables

Name	Declaration	Description
copyright	public static final String copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.
PRIORITY	public static final String PRIORITY	Priority symbol
WILDCARD	public static final String WILDCARD	Wildcard symbol

Figure 26B

2602

Constructors

Name	Declaration	Description
Transporter	public Transporter()	Default constructor.

Figure 26C

Transporter
2604

Methods

Name	Declaration	Description
addDestinationListener	public void addDestinationListener(Object major, Destination destination)	Add the Destination using the given major code. If the destination is present with the same major don't re-add it – only one major/destination pair can exist. If the major is present, but the destination isn't, add the destination to the list of other destinations with the same key. If the key isn't present, store it and then add the new destination. If the destination is disabled, do nothing.
clear	public void clear()	For each RequestEvent not started, a RequestException will be thrown and the internal data structures will be emptied including RequestEvent queues and listeners.
exit	public void exit()	For each RequestEvent not started, a RequestException will be thrown and the internal data structures will be emptied including RequestEvent queues and listeners. All variable references will be set to null.
getDestinations	public synchronized Vector getDestinations()	Return a Vector of all Destinations currently registered.
getDestinations	public Vector getDestinations(Object major)	Return a Vector of the Destinations currently registered for the given major code
getJTCs	public Vector getJTCs()	Return allocated JTC objects. By default, return the Destinations.
getMajorCodes	public Vector getMajorCodes()	Return a Vector of the registered major codes
init	public void init()	Initialize the transporter. By default, do nothing.
isEnabled	public boolean isEnabled()	Is this Transporter enabled or disabled? A Transporter that is disabled will not process an RequestEvents but will throw RequestExceptions.
isTagging	public boolean isTagging()	Is this Transporter tagging RequestEvents?
processDestinations	protected void processDestinations(RequestEvent request, Vector currentDestinations) throws RequestException	Given a RequestEvent and a Vector of destinations, call each Destination in FIFO/FEFR order. If tagging is enabled, then append a status tag to the RequestEvent.
removeDestinationListener	public void removeDestinationListener(Object major, Destination d)	Remove the destination using the given major. If the destination is not present, do nothing. If the destination is present, just remove it. If it was the last destination, remove all references to the major code.
requestEventPerformed	public void requestEventPerformed(RequestEvent request) throws RequestException	Submit a synchronous request. For each Destination that is listening for the current family of RequestEvents (the major code), send the RequestEvent to the Destination for processing. If there is a problem, throw a RequestException. Continue processing the RequestEvent as long as a RequestException is not thrown by a Destination and the RequestEvent is not consumed. If tagging is enabled, then append a status tag to the RequestEvent. Destinations are process in the following FIFO order: 1- All using "!" (priority). 2- All using a major code. 3-All using "*".
requestEventPerformed	public void requestEventPerformed(RequestEvent request, RequestResponseListener caller) throws RequestException	Submit an asynchronous request. See the synchronous requestEventPerformed for more information.
setEnabled	public void setEnabled(boolean toggle)	Enabled or disable the Transporter. A disabled Transporter will throw RequestExcpetions if accessed via requestEventPerformed.
setRequestTagging	public void setRequestTagging(boolean toggle)	Stop or start the tagging of Requests.
toString	public String toString()	Return the String Transporter plus the number of registered Destinations.

Transporter.processDestinations(RequestEvent, Vector):AUS8-1999-0693

```
/**
 * Given a RequestEvent and a Vector of destinations, call each Destination
 * in FIFO/FEFR order.
 * <p>
 * If tagging is enabled, then append a status tag to the RequestEvent.
 * @exception RequestException if the Request can't be submitted
 */
protected void processDestinations(RequestEvent request, Vector currentDestinations) throws RequestException {
      if (!enabled) {
            throw new RequestException("Transporter disabled");
      }
      if (currentDestinations == null)
            return;

      /* process FIFO/FEFR */
      Destination d = null;
      int size = currentDestinations.size();
      for (int i = 0; !request.isConsumed() && i < size; i++) {
            d = (Destination) currentDestinations.elementAt(i);
            d.requestEventPerformed(request);
            /* Try to tag the request */
            if (tagging)
                  request.setStatus(request.getStatus() + d);
      }
}
```

2604

Figure 26D

Transporter.requestEventPerformed(RequestEvent):AUS8-1999-0693

```
/**
 * Submit a synchronous request. For each Destination that is listening for
 * the current family of RequestEvents (the major code), send the RequestEvent
 * to the Destination for processing.  If there is a problem, throw
 * a RequestException. Continue processing the RequestEvent as long
 * as a RequestException is not thrown by a Destination and the RequestEvent
 * is not consumed.
 * <p>
 * If tagging is enabled, then append a status tag to the RequestEvent.
 *<p>
 * Destinations are process in the following FIFO order:
 * 1- All using "!" (priority).
 * 2- All using a major code.
 * 3- All using "*".
 *<p>
 * @exception RequestException if the Request can't be submitted
 */
public void requestEventPerformed(RequestEvent request) throws RequestException {
      if (!enabled) {
            throw new RequestException("Transporter disabled");
      }

      /* Try to tag the request */
      if (tagging)
            request.setStatus(request.getStatus() + "[Transporter]");

      /* Process PRIORITY, major and then WILDCARD destinations */
      processDestinations(request, getDestinations(PRIORITY));
      processDestinations(request, getDestinations(request.getMajor()));
      processDestinations(request, getDestinations(WILDCARD));
}
```

2606

Figure 26E

2608

```
/**
 * Submit an asynchronous request. See the synchronous
 * requestEventPerformed for more information.
 */
public void requestEventPerformed(RequestEvent request,
RequestResponseListener caller) throws RequestException {
        if (!enabled) {
                throw new RequestException("Transporter disabled");
        }
        if (tagging)
                request.setStatus(request.getStatus() +
                        "[Transporter async.]");

        //start an inner classs thread
        TransporterThread t = new TransporterThread(request, caller);
        runningThreads.put(request, t);
        t.start();
}
```

Figure 26F

Transporter.TransporterThread:AUS8-1999-0693

```
        /**
        * Private class to handle executions of submits() on another
thread.
        **/
        private class TransporterThread extends Thread {
                /**
                * The current request
                **/
                private RequestEvent request;

                /**
                * The caller of submit that we will call back
                **/
                private RequestResponseListener caller;


                /**
                * Create a transporter thread
                **/
                public TransporterThread(RequestEvent request,
RequestResponseListener caller) {
                        super();
                        this.request = request;
                        this.caller = caller;
                }
                /**
                * Just call the synchronous version of
requestEventPerformed()
                **/
                public void run() {
                        try {
                                requestEventPerformed(request);
                                caller.requestResponse(request);
                        } catch (RequestException yikes) {
                                caller.requestException(yikes);
                        } finally {
                                runningThreads.remove(request);
                        }
                }
        }
```

2610

Figure 26G

Figure 27A

Destination

2700

Variables

Name	Declaration	Description
copyright	public static final String _copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.

Figure 27B

2702

Methods

Name	Declaration	Description
getTimeout	public abstract long getTimeout()	Invoked to return the timeout value.
requestEventPerformed	public abstract void requestEventPerformed (RequestEvent request) throws RequestException	Invoked to process a RequestEvent.
setTimeout	public abstract void setTimeout(long timeout)	Invoked to set the timeout value in ms.

Figure 28A

DestinationImpl

2800

Variables

Name	Declaration	Description
copyright	public static final String copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.

Figure 28B

2802

Constructors

Name	Declaration	Description
DestinationImpl	public DestinationImpl()	Default constructor.

Figure 28C

2804

Methods

Name	Declaration	Description
clear	public void clear()	By default, do nothing.
exit	public void exit()	By default, do nothing.
getJTCs	public Vector getJTCs()	By default, do nothing.
getTimeout	public long getTimeout()	Return the timeout value.
init	public void init()	By default, do nothing.
isEnabled	public boolean isEnabled()	Is the Destination enabled?
requestEventPerformed	public void requestEventPerformed(RequestEvent request) throws RequestException	A RequestEvent has arrived. If not enabled, throw and exception. Subclasses can call this method first to see if processing should continue.
setEnabled	public void setEnabled(boolean enable)	Enable or disable the Destination. A Destination that is called when disable will throw a RequestException. By default, record it.
setTimeout	public void setTimeout(long timeout)	Set the timeout value. By default, record it.
toString	public String toString()	Returns a String that represents the value of this object which is the class name and the timeout value.

RemoteDestination.requestEventPerformed(RequestEvent):AUS8-1999-0704

2806

```
/**
 * Process request event.
 *
 * <P>PRE:  None
 * <P>POST: None
 *
 * @param request the RequestEvent object to be processed.
 * @exception RequestException if there was an error during the
 *                             processing of the event.
 */
public void requestEventPerformed(RequestEvent request) throws
RequestException {
      try {
            Method method = null;
            if (session == null) {
                  // get home interface.
                  Context ctxt = getInitialContext();
                  Object home = ctxt.lookup(request.getMajor() +
"SessionHome");

                  method = home.getClass().getMethod("create", null);
                  session = method.invoke(home, null);
            }

            // get method on home object and invoke it.
            method = session.getClass().getMethod(request.getMinor(),
                  new Class[] {Object.class});
            request.setData(method.invoke(session, new Object[]
{request.getData()}));

            if (request.getMinor().equals("remove")) {
                  session = null;
            }
      } catch (InvocationTargetException te) {
            throw new RequestException(te.getTargetException());
      } catch (Throwable t) {
            throw new RequestException(t);
      }
}
```

Figure 28D

Figure 29A

Factory

2900

Variables

Name	Declaration	Description
copyright	public static final String copyright	(c) International Business Machines Inc., 1997 1998 1999. All rights reserved.

Figure 29B

2902

Methods

Name	Declaration	Description
list	public static void list()	Show the contents of the singletons
newInstance	public static Object newInstance(String classname) throws ClassNotFoundException, InstantiationException, IllegalAccessException	Given a class name, create it and return it.
newInstance	public static Object newInstance(String classname, String key, boolean singleton) throws ClassNotFoundException, InstantiationException, IllegalAccessException	Given a class name, create the object and return it. If you want to create a singleton (true), then check to see if the object was already created and if so, return it. The class name is not used as the key but the "key" parameter is.
newInstance	public static Object newInstance(String classname, boolean singleton) throws ClassNotFoundException, InstantiationException, IllegalAccessException	Given a class name, create the object and return it. If you want to create a singleton (true), then check to see if the object was already created and if so, return it. Use the class name as the key.
newInstances	public static Vector newInstances(String classnames[]) throws ClassNotFoundException, InstantiationException, IllegalAccessException	Given some class names, create and return a Vector of objects.
newInstances	public static Vector newInstances(String classnames[], String keys[], boolean singleton) throws ClassNotFoundException, InstantiationException, IllegalAccessException	Given some class names, create and return a Vector of objects. If you want singleton objects system wide, then if any of the classes were already created, return them, otherwise, create the new ones, remember them and return them. The class names are not used as the keys but the "keys" parameter are.
newInstances	public static Vector newInstances(String classnames[], boolean singleton) throws ClassNotFoundException, InstantiationException, IllegalAccessException	Given some class names, create and return a Vector of objects. If you want singleton objects system wide, then if any of the classes were already created, return them, otherwise, create the new ones, remember them and return them. Use the class name as the key
removeInstance	public static void removeInstance(String key) throws ClassNotFoundException, InstantiationException, IllegalAccessException	Given a class key, clear the reference to it.
removeInstances	public static void removeInstances(String keys[]) throws ClassNotFoundException, InstantiationException, IllegalAccessException	Given some class keys, clear the references.

Interface com.ibm.jtc.JTC

3000

FIGURE 30A

Variables		
Name	Declaration	Description
_copyright	Public static final String _copyright	International Business Machines Inc., 19997 1998 1999. All rights reserved.
_version	Public static final String version	
_author	Public static final String _author	
_email	Public static final String _email	

3002

FIGURE 30B

Methods		
Name	Declaration	Description
clear	Public abstract void clear()	Invoked to indicate that all memory allocations should be cleaned up. This includes removing listeners and flushing any lists (vectors or hashtables). A JTC object that has been cleared can be reused.
exit	public abstract void exit()	Invoked to indicate that all memory allocations should be cleaned up. This includes removing listeners and flushing any lists (vectors or hashtables). It also includes setting all variable references to null. A JTC object that has been cleared cannot be reused.
getJTCs	Public abstract Vector getJTCs()	Invoked to get a Vector of all JTC objects that this JTC object has created. For example, a Transporter will at least return all of its Destinations. This is a very powerful mechanism. It allows us to get a reference to all primary objects in the JTC application and manipulate them according to the JTC methods, or by casting them to more specific classes or interfaces and manipulating them. Examples usage includes non code intrusive tracing, debugging, logging, profiling, etc.
init	Public abstract void init()	Invoked to initialize the JTC object. The object should be ready for operation.
isEnabled	Public abstract boolean isEnabled()	Invoked to determine if the JTC object is enabled.
setEnabled	Public abstract void setEnabled(boolean enable)	Invoked to enable or disable the JTC object.
toString	Public abstract String toString()	Invoked to get a String representation of the JTC object.

Figure 31

View Controller

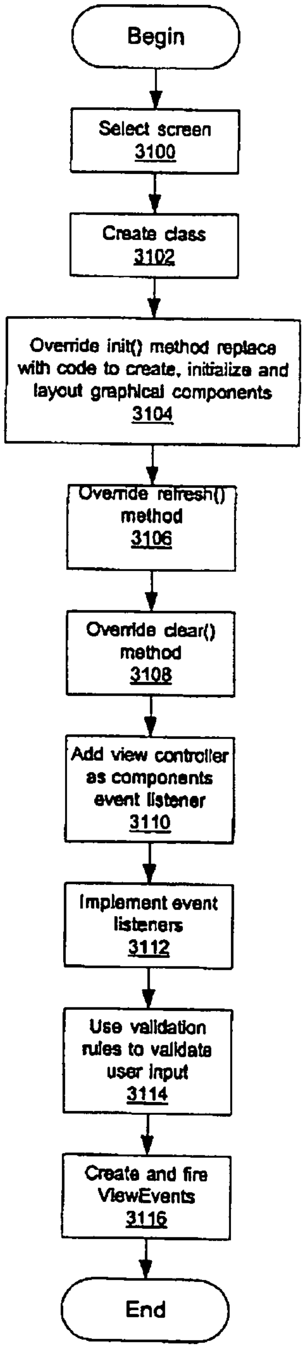

Begin
Select screen 3100
Create class 3102
Override init() method replace with code to create, initialize and layout graphical components 3104
Override refresh() method 3106
Override clear() method 3108
Add view controller as components event listener 3110
Implement event listeners 3112
Use validation rules to validate user input 3114
Create and fire ViewEvents 3116
End

Figure 32

Create Validation Rule

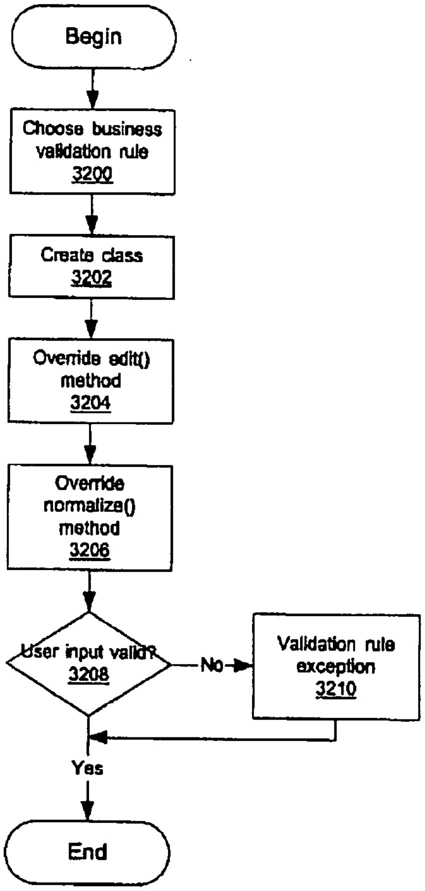

Begin
Choose business validation rule 3200
Create class 3202
Override edit() method 3204
Override normalize() method 3206
User input valid? 3208
No
Validation rule exception 3210
Yes
End

Figure 33

Create a ViewEvent

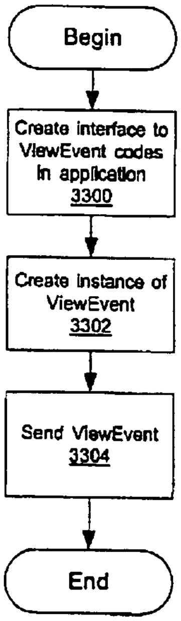

Begin
Create interface to ViewEvent codes in application 3300
Create instance of ViewEvent 3302
Send ViewEvent 3304
End

Figure 34

Create ApplicationMediator

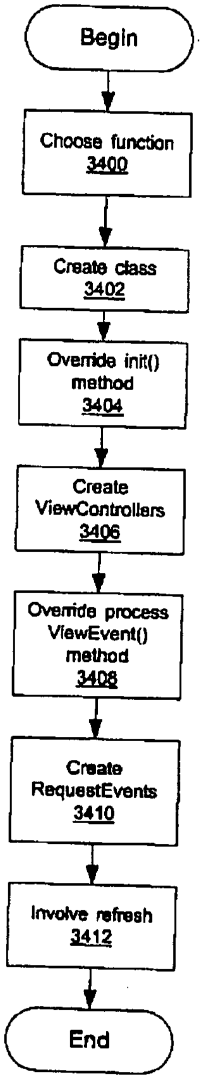

Begin
Choose function 3400
Create class 3402
Override init() method 3404
Create ViewControllers 3406
Override process ViewEvent() method 3408
Create RequestEvents 3410
Involve refresh 3412
End

Figure 35

Create RequestEvent

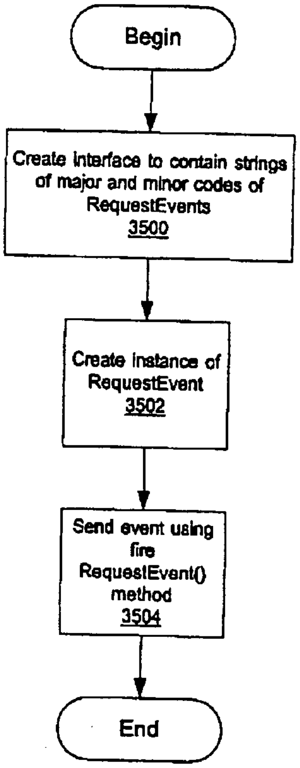

Begin
Create interface to contain strings of major and minor codes of RequestEvents 3500
Create instance of RequestEvent 3502
Send event using fire RequestEvent() method 3504
End

Figure 36

Create a Destination

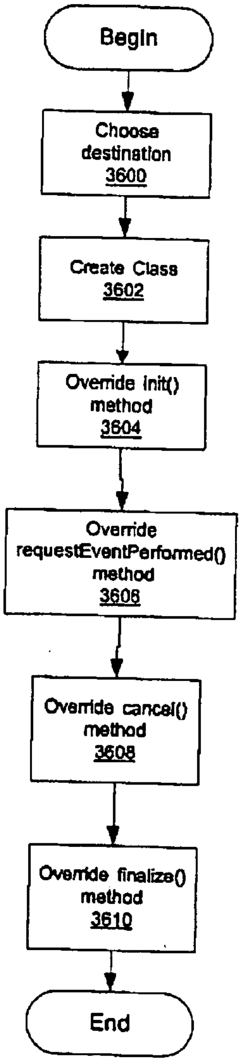
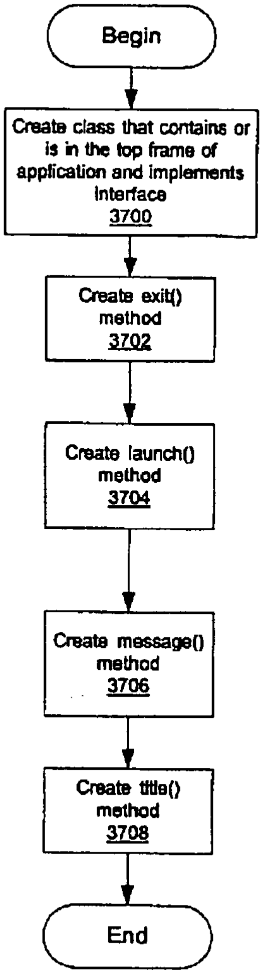

Begin
Choose destination 3600
Create Class 3602
Override init() method 3604
Override requestEventPerformed() method 3606
Override cancel() method 3608
Override finalize() method 3610
End

Figure 37

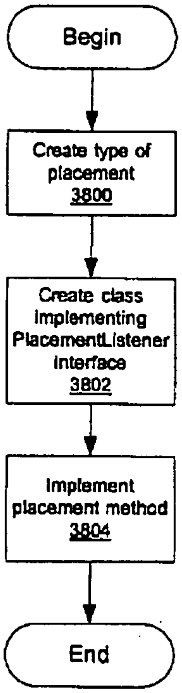
Create TopListener



Begin
Create class that contains or is in the top frame of application and implements interface 3700
Create exit() method 3702
Create launch() method 3704
Create message() method 3706
Create title() method 3708
End

Figure 38

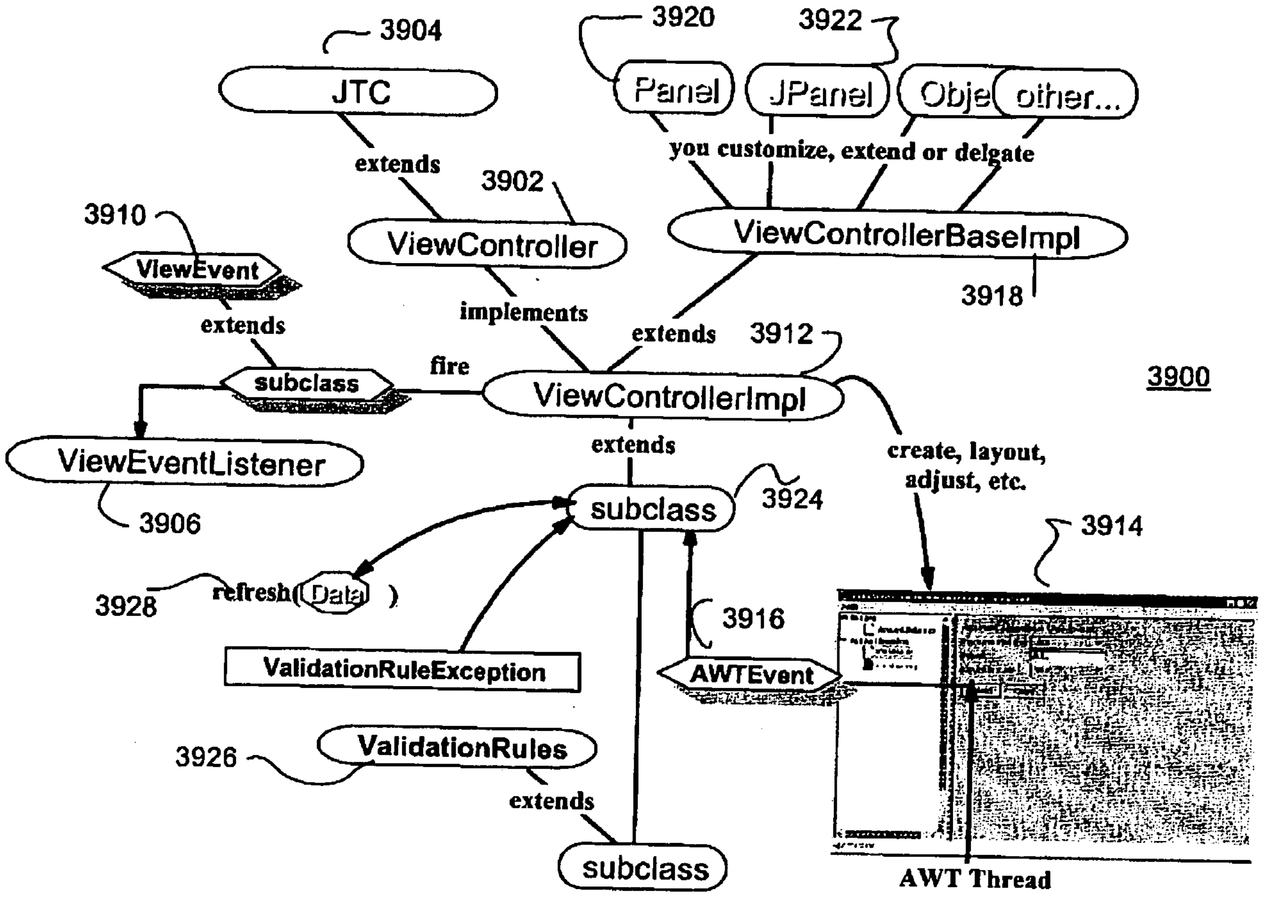
Create PlacementListener



Begin
Create type of placement 3800
Create class implementing PlacementListener interface 3802
Implement placement method 3804
End

Figure 39

ViewController runtime



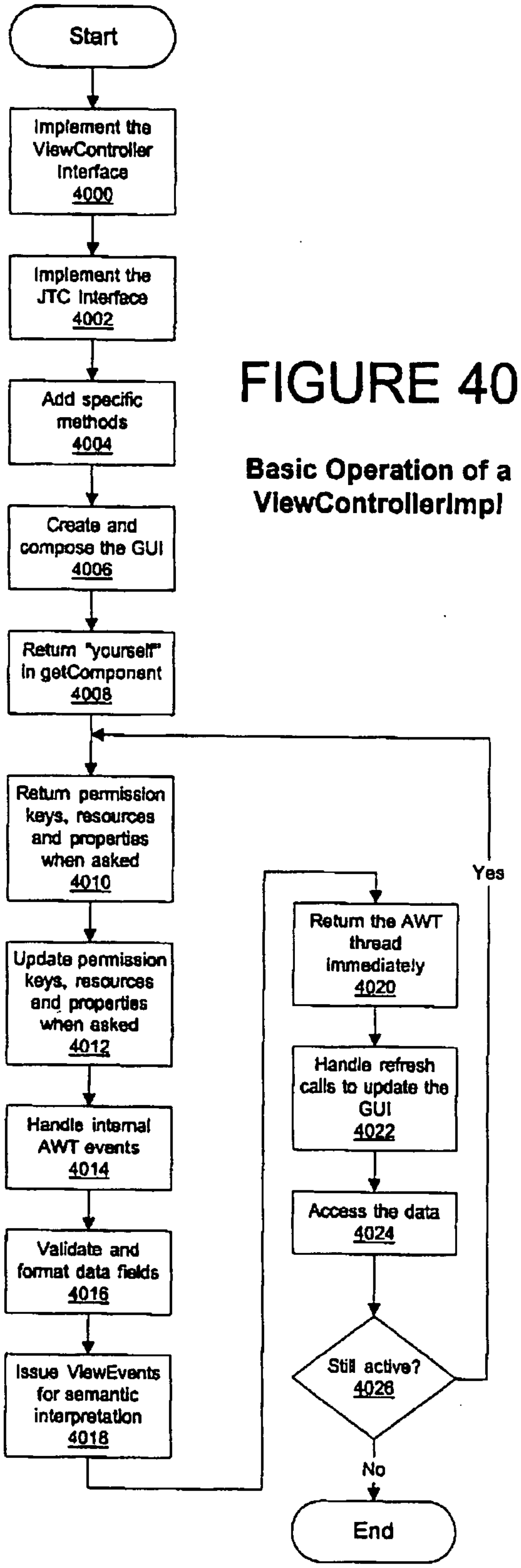
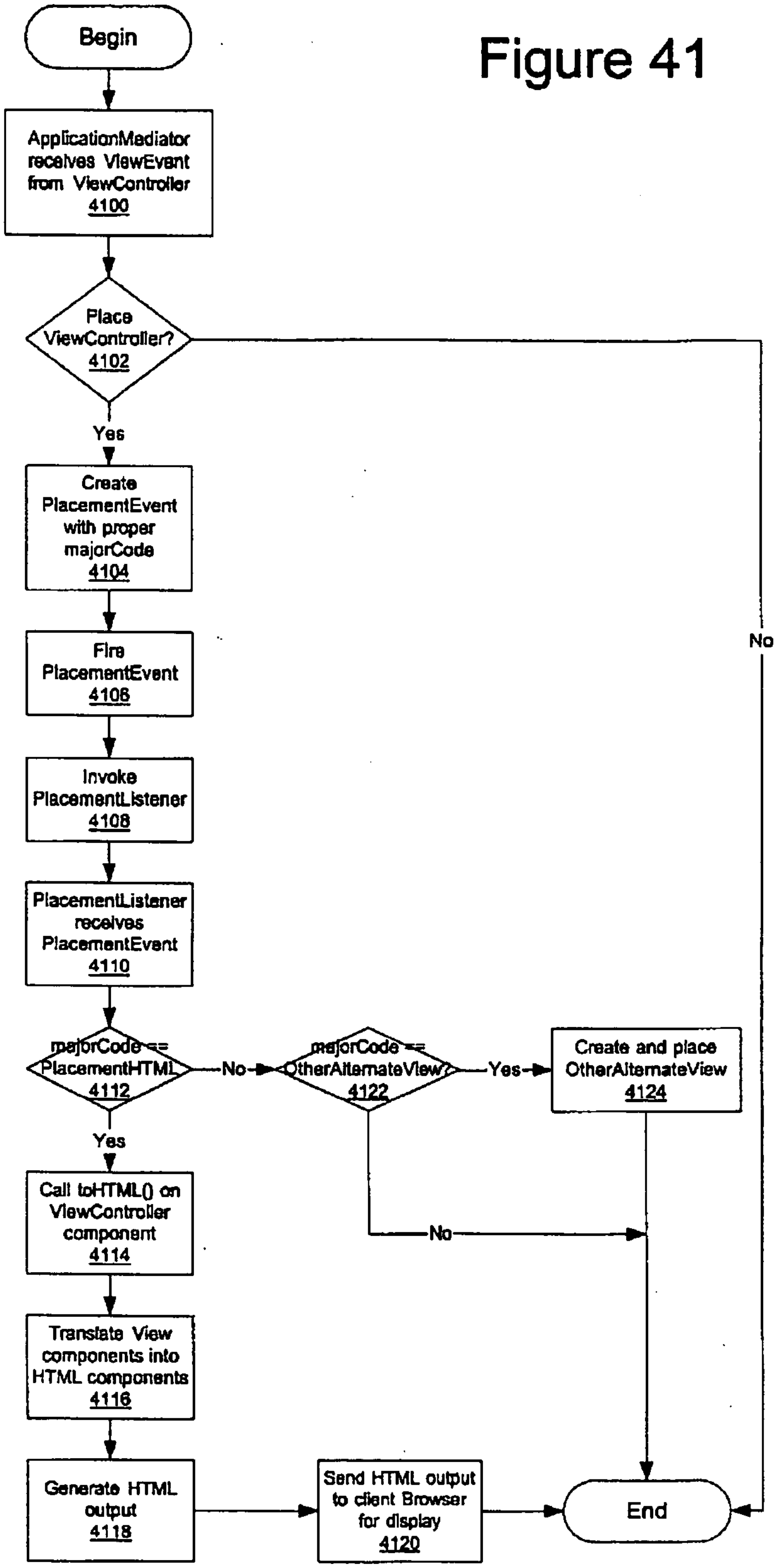

Start
Implement the ViewController Interface 4000
Implement the JTC Interface 4002
Add specific methods 4004
Create and compose the GUI 4006
Return "yourself" in getComponent 4008
Return permission keys, resources and properties when asked 4010
Update permission keys, resources and properties when asked 4012
Handle internal AWT events 4014
Validate and format data fields 4016
Issue ViewEvents for semantic interpretation 4018
Return the AWT thread immediately 4020
Handle refresh calls to update the GUI 4022
Access the data 4024
Still active? 4026
Yes
No
End

FIGURE 40

Basic Operation of a ViewControllerImpl

Figure 41


Begin
ApplicationMediator receives ViewEvent from ViewController 4100
Place ViewController? 4102
Yes
No
Create PlacementEvent with proper majorCode 4104
Fire PlacementEvent 4106
Invoke PlacementListener 4108
PlacementListener receives PlacementEvent 4110
majorCode == PlacementHTML 4112
No
majorCode == OtherAlternateView? 4122
Yes
Create and place OtherAlternateView 4124
No
Yes
Call toHTML() on ViewController component 4114
Translate View components into HTML components 4116
Generate HTML output 4118
Send HTML output to client Browser for display 4120
End

ViewEvent and ViewListener Usage

FIGURE 42

```
→Usage from a ViewController
  public void actionPerformed(ActionEvent e){
      if (e.getSource() == nextButton) {
              ViewEvent ve = new ViewEvent (this);
              ve.setMajor(ViewEvent.NEXT);
              fireViewEvent(ve); //notify
      ViewEvent listener
              return;
      }
   }
```

FIGURE 43

```
→Usage from ViewListener (i.e. ApplicationMediator)
  //add myself as a listener
  customerDetailsViewController.addViewListener(this);

  //later, we are called back on this method to handle the
ViewEvent
  processViewEvent (ViewEvent event) {
     //do something
     switch (event.getMajor()) {
        case ViewEvent.NEXT: //...
        break;
        case ViewEvent.OK: //...
        break;
     }
  }
```

Major and/or minor codes

FIGURE 44

➡ Pre-defined major codes- A subclass can define others.

- // system
  - OK DONE OPEN CLOSE CANCEL EXIT FILE SAVE SAVEAS ERROR WARNING RETURN LOAD NOTIFY NOTIFY2 INFO SETUP PRINT LOGIN LOGOUT ENABLE DISABLE
- // status
  - TITLEMESSAGE STATUSMESSAGE ERRORMESSAGE SUGGESTIONMESSAGE
- // navigational
  - NEXT PREVIOUS FIRST LAST START BEGIN END PAUSE STOP RESTART SUBMIT BACKSPACE INSERT HOME PGUP PGDN LEFT RIGHT UP DOWN
- // live
  - FAST MEDIUM SLOW RUN DELAY WAIT TIMER ON OFF HIGH LOW
- // data related
  - LIST MORE ADD DELETE MODIFY NEW EDIT COPY CUT PASTE UNDO REMOVE PLUS MINUS INCREMENT DECREMENT CHANGED FILL EMPTY READY VIEW DETAILS READ WRITE UPDATE REFRESH
- // assit related
  - SEARCH FIND HELP HINT TRAIN TEACH SUGGEST
- //sub options related
  - A B C D E F OPTION CHOOSE
- // test values
  - TRACE UNTRACE DEBUG UNDEBUG LOG UNLOG HOOK UNHOOK
- // ibm values
  - TEAM WIN EXECUTE

ValidationRules Usage

➡ Examples:

```
edit("123456")            ->  $1234.56
normalize("$1234.56")     ->  123456
edit("12345x")            ->  ValidationRuleException
```

➡ edit

```
//validate and re-display
String value = textfield.getText();
try {
    result = SocialSecurity.edit(value);
}
catch (ValidationRuleException yikes) {
    //...
    return;
}
textField.setText(value);
```

FIGURE 45

➡ normalize

```
//validate and update the data objects
String value = textfield.getText();
try {
    result = SocialSecurity.normalize(value);
}
catch (ValidationRuleException yikes) {
    //message box ...
    return;
}
dataObject.setText(value);
```

FIGURE 46

ValidationRules Usage

FIGURE 47

➡ Example Chaining

```
//each rule
String range = "com.xyz.jtc.RangeChecker";
String money = "com.xyz.jtc.AccountMoney";

//build the chain of rules
String[] rules = {range, money};

//get the value to validate
String value = textField.getText();

try {
    value = applyEdits(rules, input);
}
catch (ValidationRuleException ouch) {
    //...
}

//the value is validated and formatted, redisplay
textField.setText(value);
```

ViewControllerBaseImpl

➡For example:

- inheritance

FIGURE 48

```
public class ViewControllerBaseImpl extends JPanel {
    public Component getComponent() {
        return this;
    }
}
```

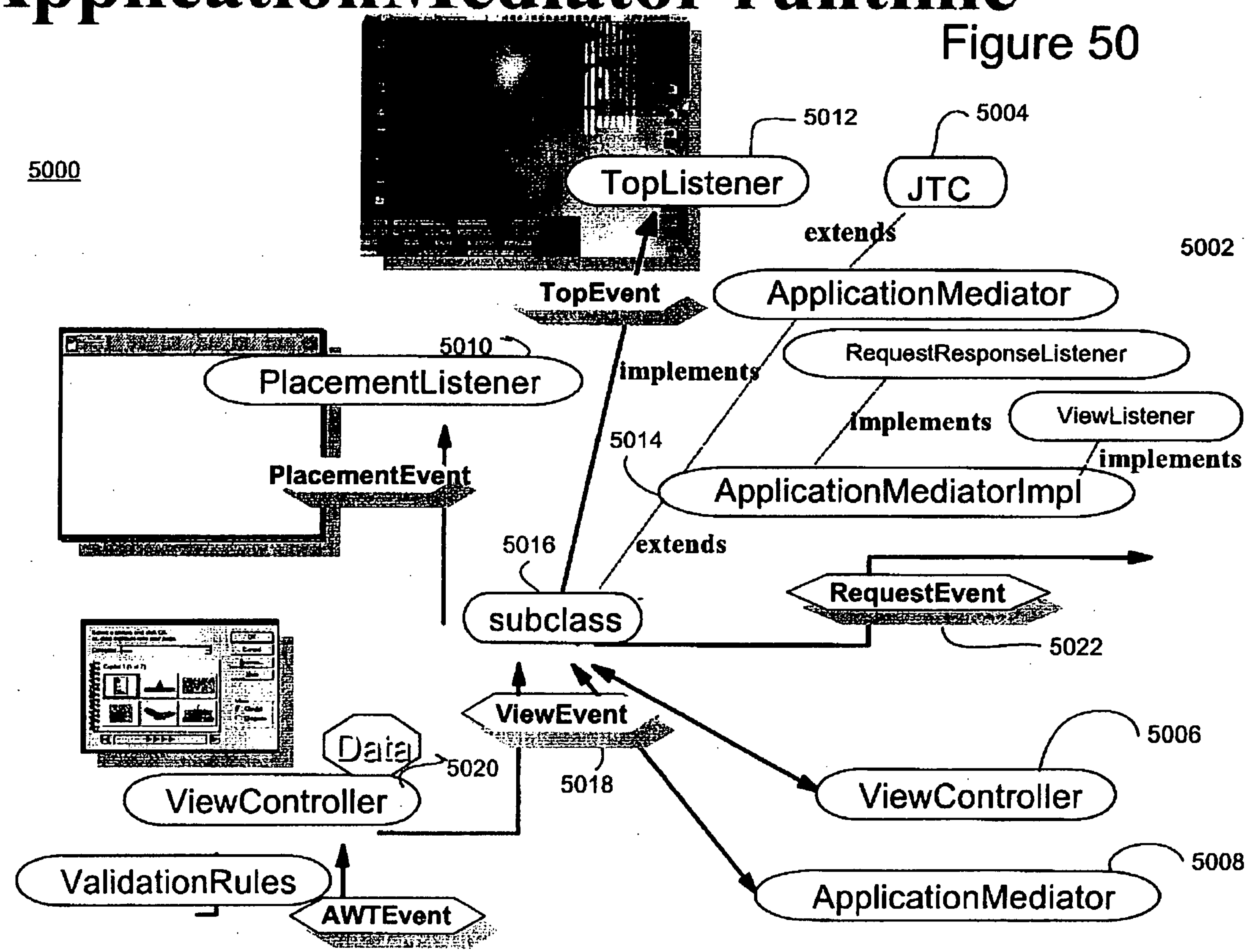
FIGURE 49

- delegation

```
public class ViewControllerBaseImpl implements ViewController
{
    XYZ xyz = new XYZ();

    public java.awt.Component getComponent() {
        return xyz;
    }

    public void setEnabled(boolean e) {
        xyz.setEnabled(e);
    }

    public void setVisible(boolean v) {
        xyz.setVisible(v);
    }
}
```

ApplicationMediator runtime

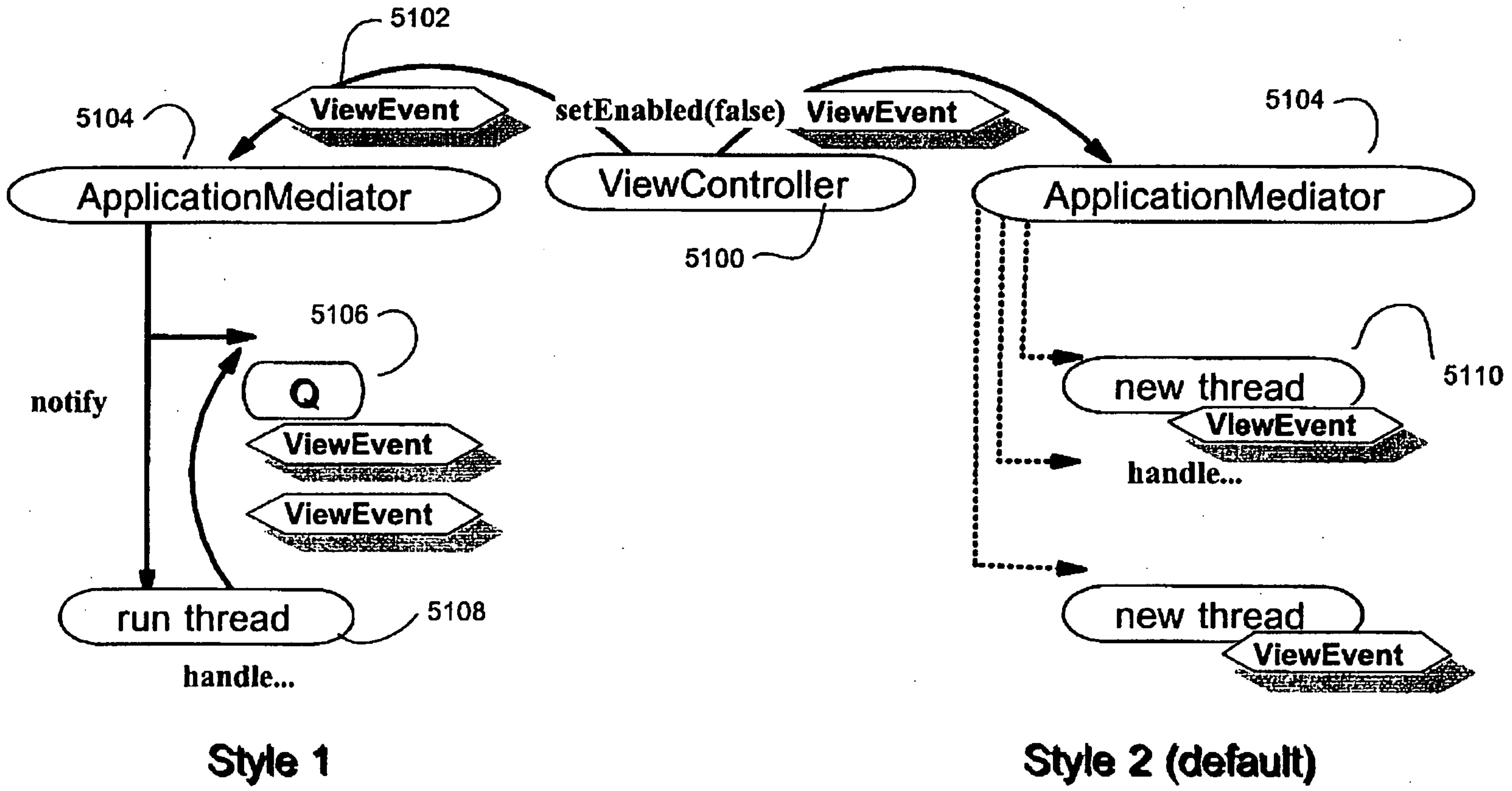
Figure 50



AWTEvent threading support

Figure 51

Style 1 - wait / Queue / notify
Style 2 - Thread dispatch
- Handles Threading Model for ViewControllers



Style 1

Style 2 (default)

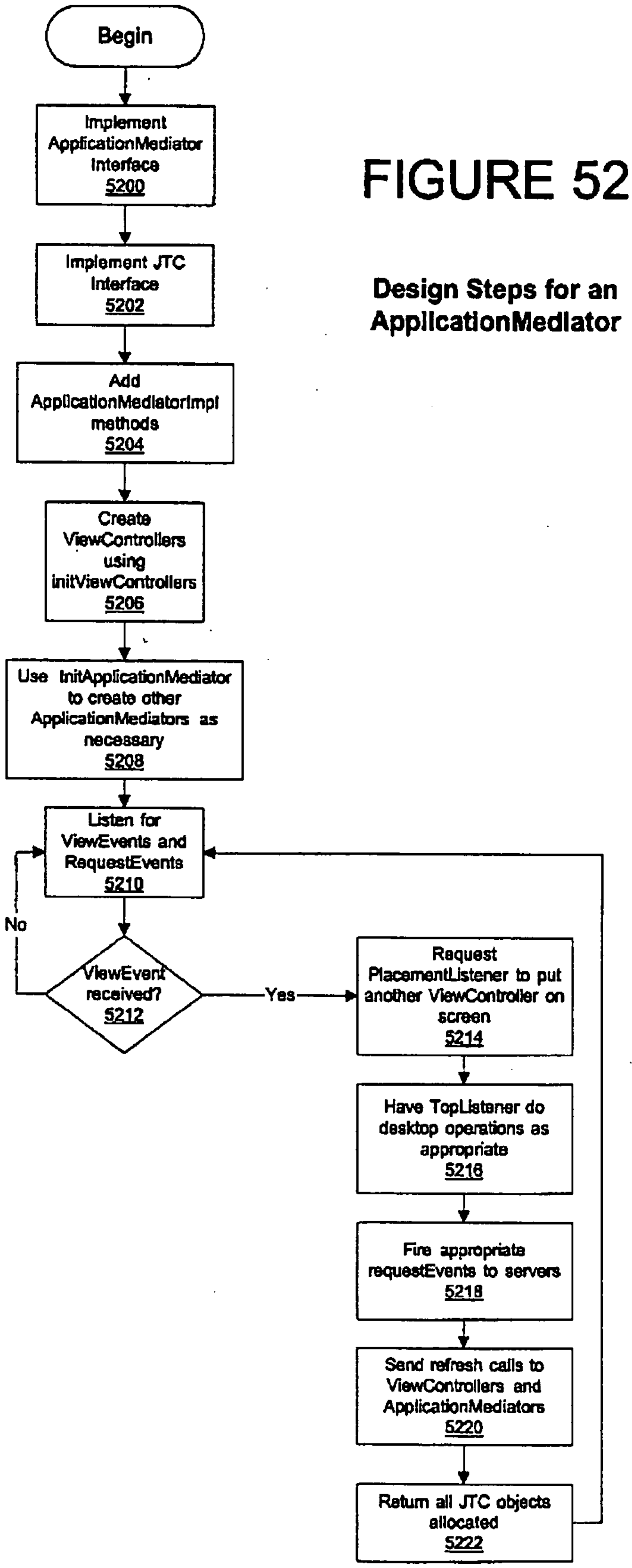
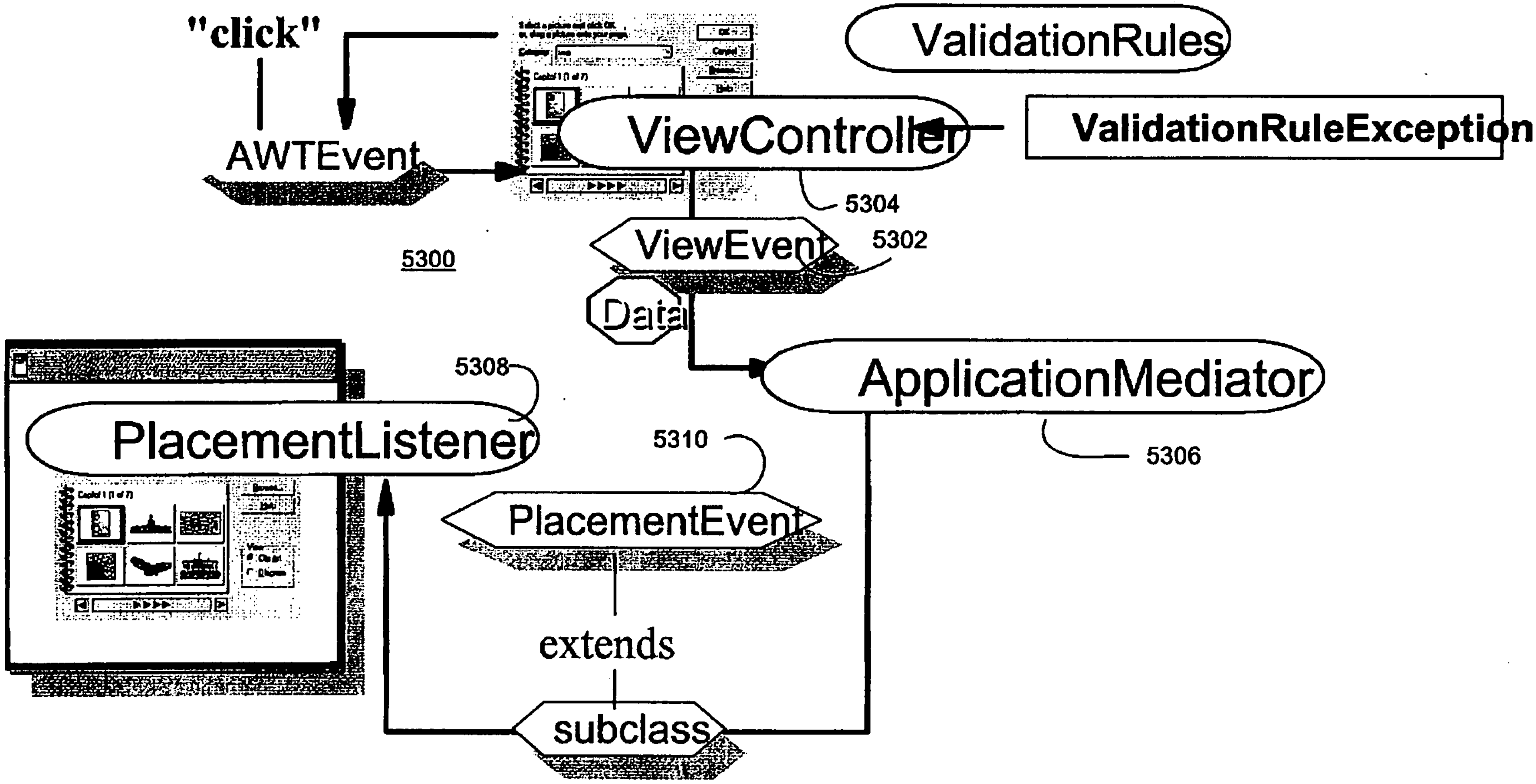

Begin
Implement ApplicationMediator Interface 5200
Implement JTC Interface 5202
Add ApplicationMediatorImpl methods 5204
Create ViewControllers using initViewControllers 5206
Use InitApplicationMediator to create other ApplicationMediators as necessary 5208
Listen for ViewEvents and RequestEvents 5210
No
ViewEvent received? 5212
Yes
Request PlacementListener to put another ViewController on screen 5214
Have TopListener do desktop operations as appropriate 5216
Fire appropriate requestEvents to servers 5218
Send refresh calls to ViewControllers and ApplicationMediators 5220
Return all JTC objects allocated 5222

FIGURE 52

Design Steps for an ApplicationMediator

Placement runtime Figure 53



Placement example

➡Usage from ApplicationMediator

FIGURE 54

```
//in an ApplicationMediator
int major = PlacementEvent.ADD;
Component component =
customerDetailsViewController.getComponent();
PlacementEvent e = new PlacementEvent(this, component, major);
firePlacementEvent(e);
```

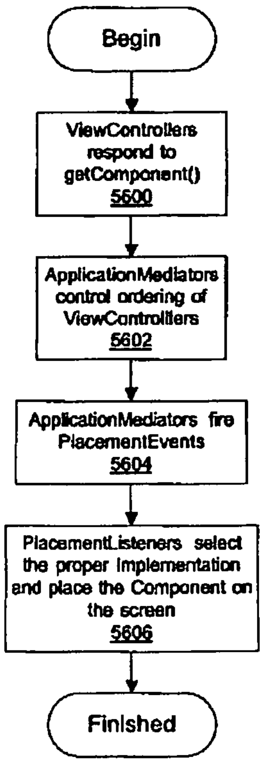
➡Usage from PlacementListener

FIGURE 55

```
public class MyProgram implements PlacementListener {
    public void placementEventPerformed(PlacementEvent e) {
         //decide based on source type
         switch (e.getMajor()) {

              case PlacementEvent.ADD:
                   if (e.getSource() instanceof PreferencesAm)
                        panel1.add("Center", e.getComponent());
                   else panel2.add("A", e.getComponent());
              break;
              case PlacementEvent.REMOVE:
                   //do something else
              break;
         }
      //etc.
      }
 }
```

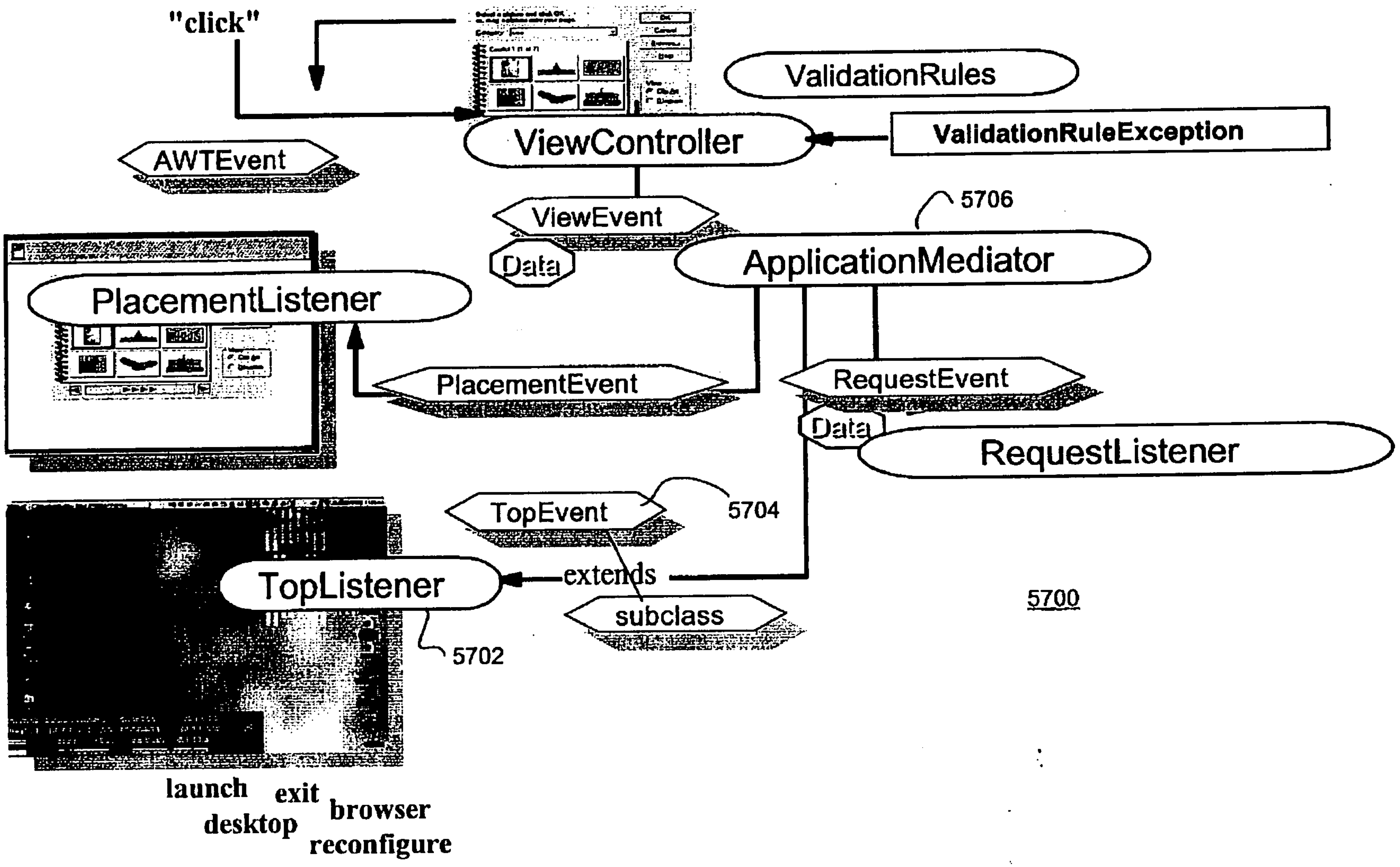
FIGURE 56

Design Steps for a PlacementEvent


Begin
ViewControllers respond to getComponent() 5600
ApplicationMediators control ordering of ViewControllers 5602
ApplicationMediators fire PlacementEvents 5604
PlacementListeners select the proper Implementation and place the Component on the screen 5606
Finished

TopListener runtime

Figure 57



TopListener example

FIGURE 58

```
//from the TopListener
ApplicationMediatorXYZ m = new ApplicationMediatorXYZ();
m.addTopListener(this);
```

FIGURE 59

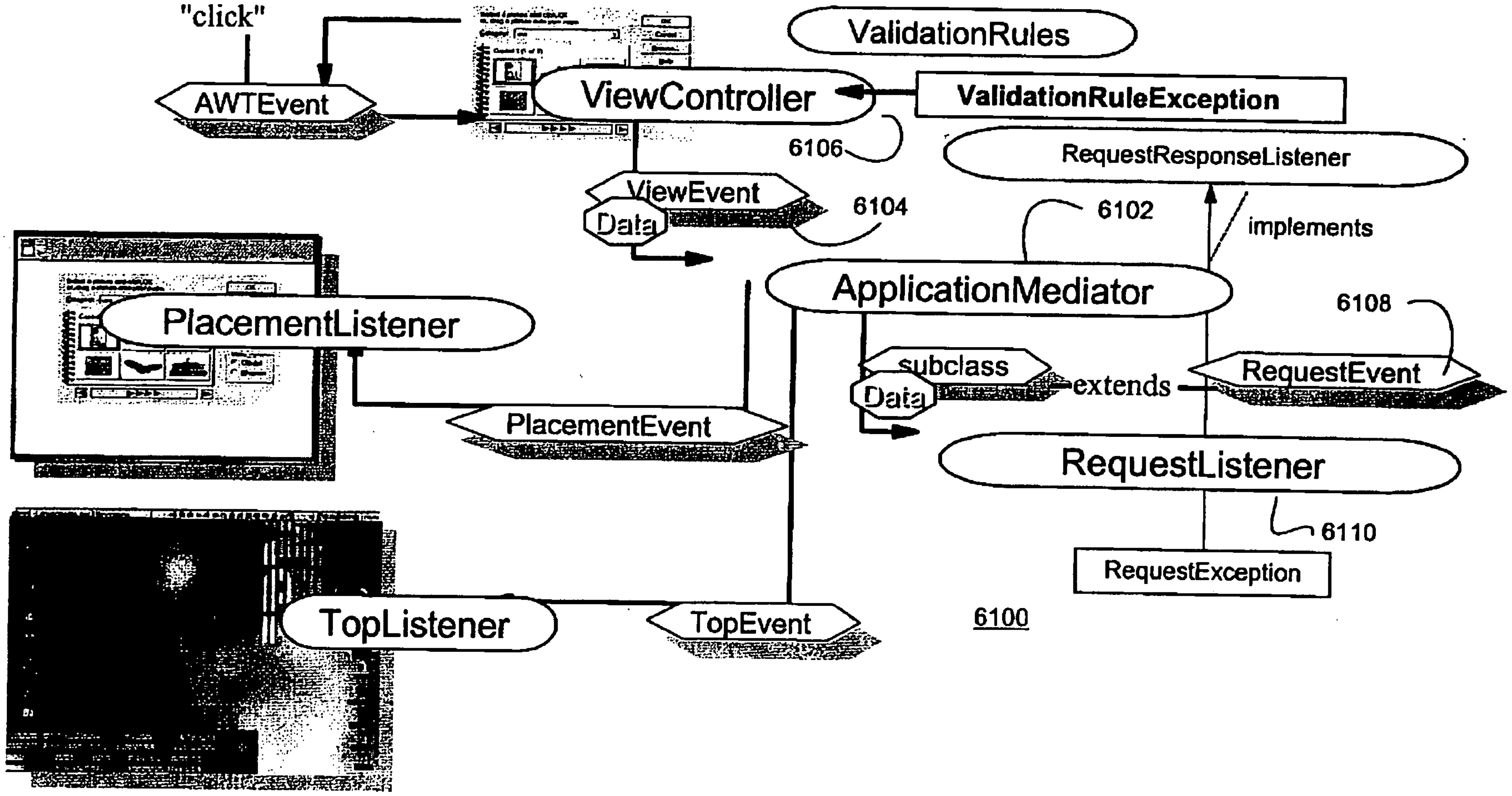
```
//in the ApplicationMediator
String status = "Loading files...";
TopEvent e = new TopEvent(this, TopEvent.STATUS, 0, status);
fireTopEvent(e);
```

FIGURE 60

```
//later in the TopListener callback
public void topEventPerformed(TopEvent e) {
    switch(e.getMajor()) {
        case STATUS:
            //access the browser
        break;
        /etc.
    }
}
```

RequestEvent runtime

Figure 61



RequestEvent example

```
//from an ApplicationMediator - create event
RequestEvent r = new RequestEvent();
r.setMajor("Loans");
r.setMinor("SubmitCustomerInfo");
```

FIGURE 62

```
//fire an asynchronous event
try {
    //asynchronous
    fireRequestEvent(this, r);
}
catch (RequestException yikes) {}
```

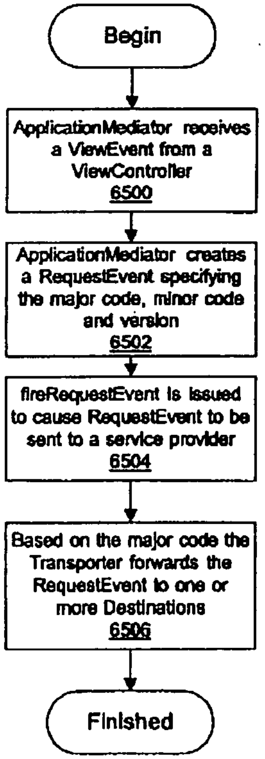
FIGURE 63

FIGURE 64

```
//later, called back with success
public void requestResponse(RequestEvent result) {
   //process response
}

//or failure
public void requestException(RequestException yikes) {
    //now what?
}
```

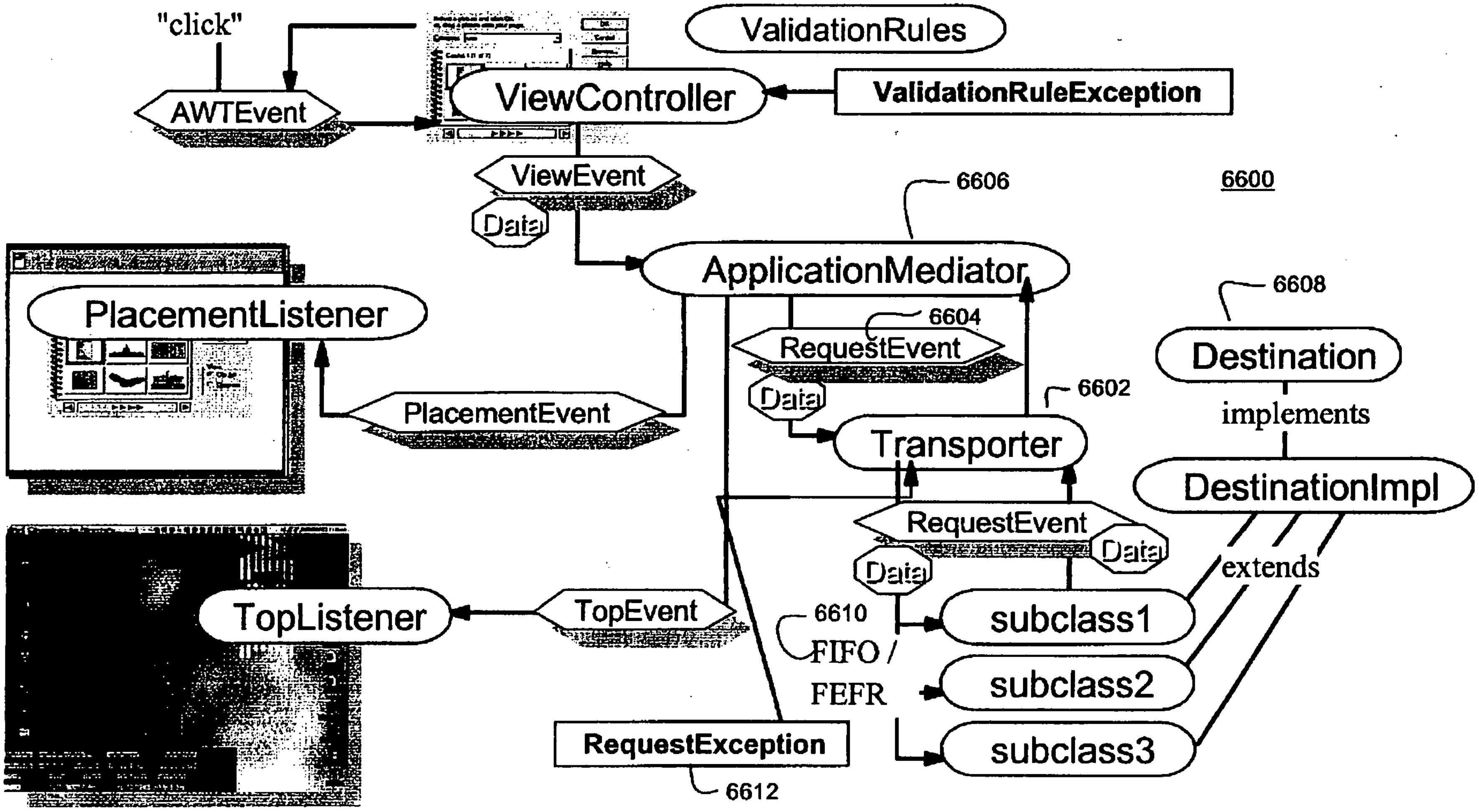
FIGURE 65

Design Steps for a RequestEvent


Begin
ApplicationMediator receives a ViewEvent from a ViewController
6500
ApplicationMediator creates a RequestEvent specifying the major code, minor code and version
6502
fireRequestEvent is issued to cause RequestEvent to be sent to a service provider
6504
Based on the major code the Transporter forwards the RequestEvent to one or more Destinations
6506
Finished

Transporter runtime

Figure 66



Transporter

- ➡ This class implements the JTC and RequestEventListener interfaces
- ➡ Its primary function is to map RequestEvents to Destinations.
  - Typically ApplicationMediators fire RequestEvents and Destinations process them
- ➡ Add a Transporter to an ApplicationMediator to listen for RequestEvents

```
Transporter t = new Transporter();
ApplicationMediator am = new ApplicationMediator();
am.addRequestListener(t);
```

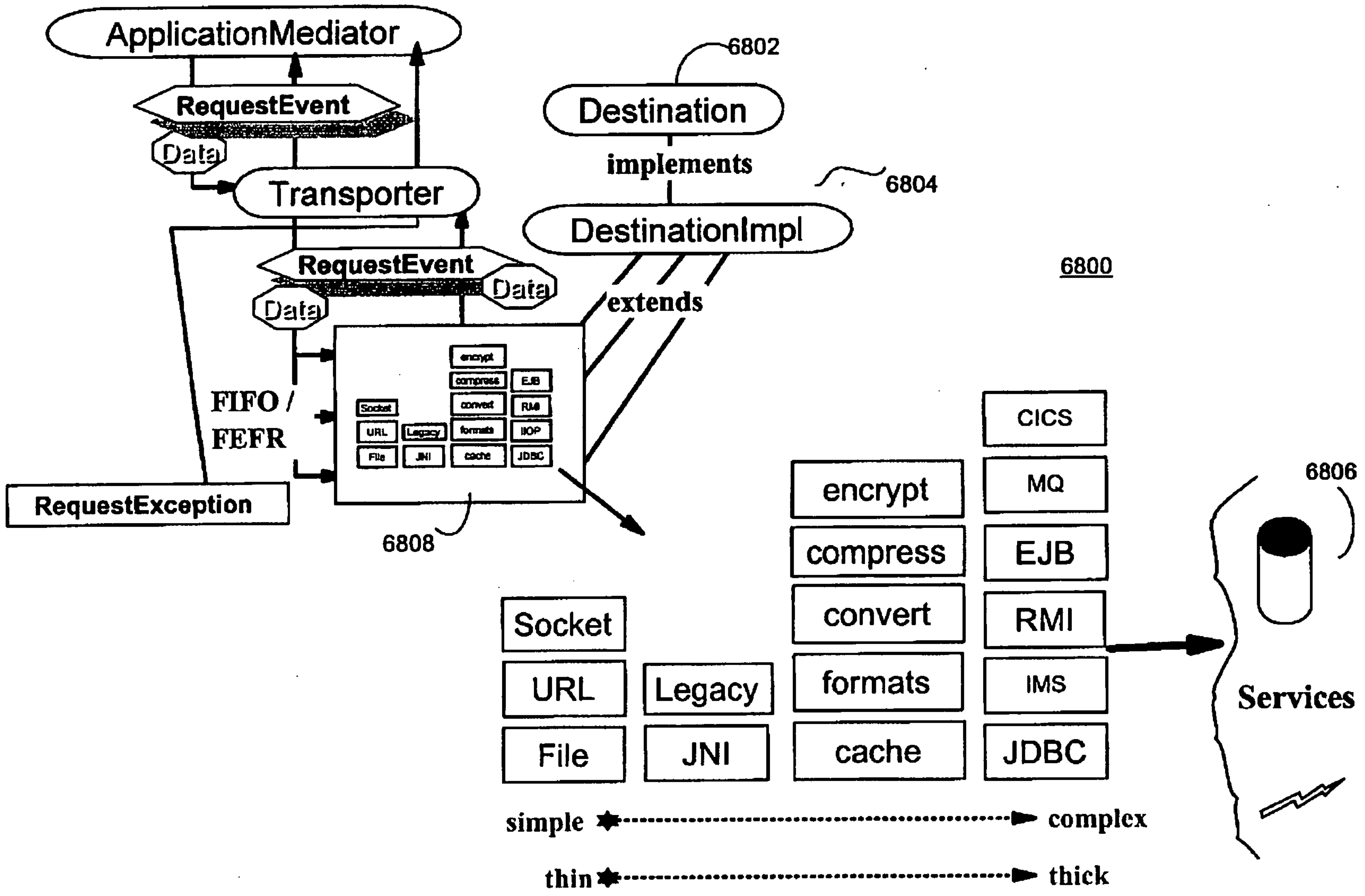
FIGURE 67

- ➡ The ApplicationMediator will fire RequestEvents

```
RequestEvent r = new RequestEvent(source, major, minor, data);
try {
     fireRequestEvent(r);
}
catch (RequestException yikes) {}
```

Destination runtime

Figure 68



Destination

- ➡ RequestEvents are identified by
  - major code - represents a family of Requests
  - minor code - represents a specific Request
- ➡ Destinations are added to the Transporter as DestinationListeners
  - specifing a major code for RequestEvents they are interested in receiving
- ➡ The destination is called when the major code of the RequestEvent matches the destination major code

```
EJBDestination d = new EJBDestination();
Transporter t = new Transporter();
String major = "Loans";
t.addDestinationListener(major, d);
```

FIGURE 69

- ➡ Multiple Destinations can listen for the same RequestEvent major code
  - processed FIFO / FESP (first in first out, first exception stop forwarding)
  - results of one Destination can be passed to the next Destination

Destinations and major codes

- ➡ Special major codes
  - wildcard
    - "*" major code indicates the Destination is interested in all and any RequestEvents
    - processed after specific major codes have been matched.
  - priority
    - "!" major code indicates the Destination is interested in all requests and should be given priority.
    - processing performed before specific major codes and wildcards
- ➡ For example

```
Transporter t = new Transporter();
t.addDestinationListener ("*",     new WildDestination());
t.addDestinationListener ("Loans", new EJBDestination());
t.addDestinationListener ("!",     new PriorityDestination()):

//later
RequestEvent r = new RequestEvent(this, "Loans", "", null);
try {
    fireRequestEvent(r);
}
catch (RequestException yikes) {}
```

FIGURE 70

- The RequestEvent "r" will be sent to PriorityDestination 1st, EJBDestination 2nd, and WildDestination() 3rd, assuming no RequestExcpeptions are thrown.

getJTCs example

FIGURE
71

```
// Recursively look at the root, find each JTC and/or AWT and hook
public void hookJTCs(JTC root) {
    Vector jtcs = null;
    try {
         jtcs = root.getJTCs();
    } catch (Exception none) { return; } // should not happen

    if (jtcs == null) return; //we are done

    int size = jtcs.size();
    for (int j = 0; j < size; j++) {
         Object current = jtcs.elementAt(j);
         if (current instanceof ApplicationMediator) {
              hookAM((ApplicationMediator) current);
         } else
              if (current instanceof ViewController) {
                   hookVC((ViewController) current);
              } else
                   if (current instanceof Transporter) {
                        hookTransporter((Transporter) current);
                   } else
                        if (current instanceof java.awt.Component) {
                             //once into AWT tree, never back to JTCs
                             hookAWTs((java.awt.Component) current);
                             continue;
                        }
         hookJTCs((JTC) jtcs.elementAt(j)); //recursive
    }
}
```

hookJTC helpers

FIGURE
72

```
/**
 * Hook the ApplicationMediator
 */
public void hookAM(ApplicationMediator am) {
    vcl.refresh("ApplicationControllers found:" + am);
    am.addViewListener(this);
    am.addRequestListener(this);
    vcl.refresh("....add as ViewListener");
    vcl.refresh("....add as RequestListener");
}

/**
 * Hook the ViewController and it's getComponent()
 */
public void hookVC(ViewController vc) {
    vcl.refresh("ViewController found:" + vc);
    vc.addViewListener(this);
    vcl.refresh("....add as ViewListener");
    hookAWTs(vc.getComponent());
}

/**
 * Hook the Transporter
 */
public void hookTransporter(Transporter transporter) {
    vcl.refresh("Transporter found:" + transporter);
    transporter.addDestinationListener("|", this);
    vcl.refresh("....add as | DestinationListener");
}
```

hookAWTs

FIGURE
73

```
// Recursively find each AWT object and hook
public void hookAWTs(Component comp) {
    if (component instanceof Container) {
          vcl.refresh("Container found:" + comp);
          Component[] comps = ((Container) comp).getComponents();
          int size = comps.length;
          for (int i = 0; i < size; i++) {
               hookAWTs(comps[i]);
          }
    }
    /* continue here since some regular Components, such as JLabels,
    * are Containers also.
    */
    if (comp instanceof Button) {
          hookAWTButton((Button) comp);
    } else
          if (comp instanceof JButton) {
               hookSwingJButton((JButton) comp);
          } else
               if (component instanceof  JTextField) {
                    hookSwingJTextField((JTextField) comp);
               }
               /*...else do over every other Bean/Component/Container
              * type possibly using reflection or a table driven
              * implementation.
             */
}
```

hookAWTs - helpers

FIGURE
74

```
/**
 * Hook the java.awt.Button
 */
public void hookAWTButton(Button button) {
    vcl.refresh("java.awt.Button found:" + button);
    button.addActionListener(this);
    vcl.refresh("....add as ActionListener");
}
/**
 * Hook the com.sun.java.swing.JButton
 */
public void hookSwingJButton(JButton button) {
    vcl.refresh("com.sun.java.swing.JButton found:" + button);
    button.addActionListener(this);
    button.addChangeListener(this);
    button.addItemListener(this);
    vcl.refresh("....add as ActionListener");
    vcl.refresh("....add as ChangeListener");
    vcl.refresh("....add as ItemListener");
}
/**
 * Hook the com.sun.java.swing.JTextField
 */
public void hookSwingJTextField(JTextField textfield) {
    vcl.refresh("com.sun.java.swing.JTextField found:" + textfield);
    textfield.addActionListener(this);
    textfield.addCaretListener(this);
    vcl.refresh("....add as ActionListener");
    vcl.refresh("....add as CaretListener");
}
```

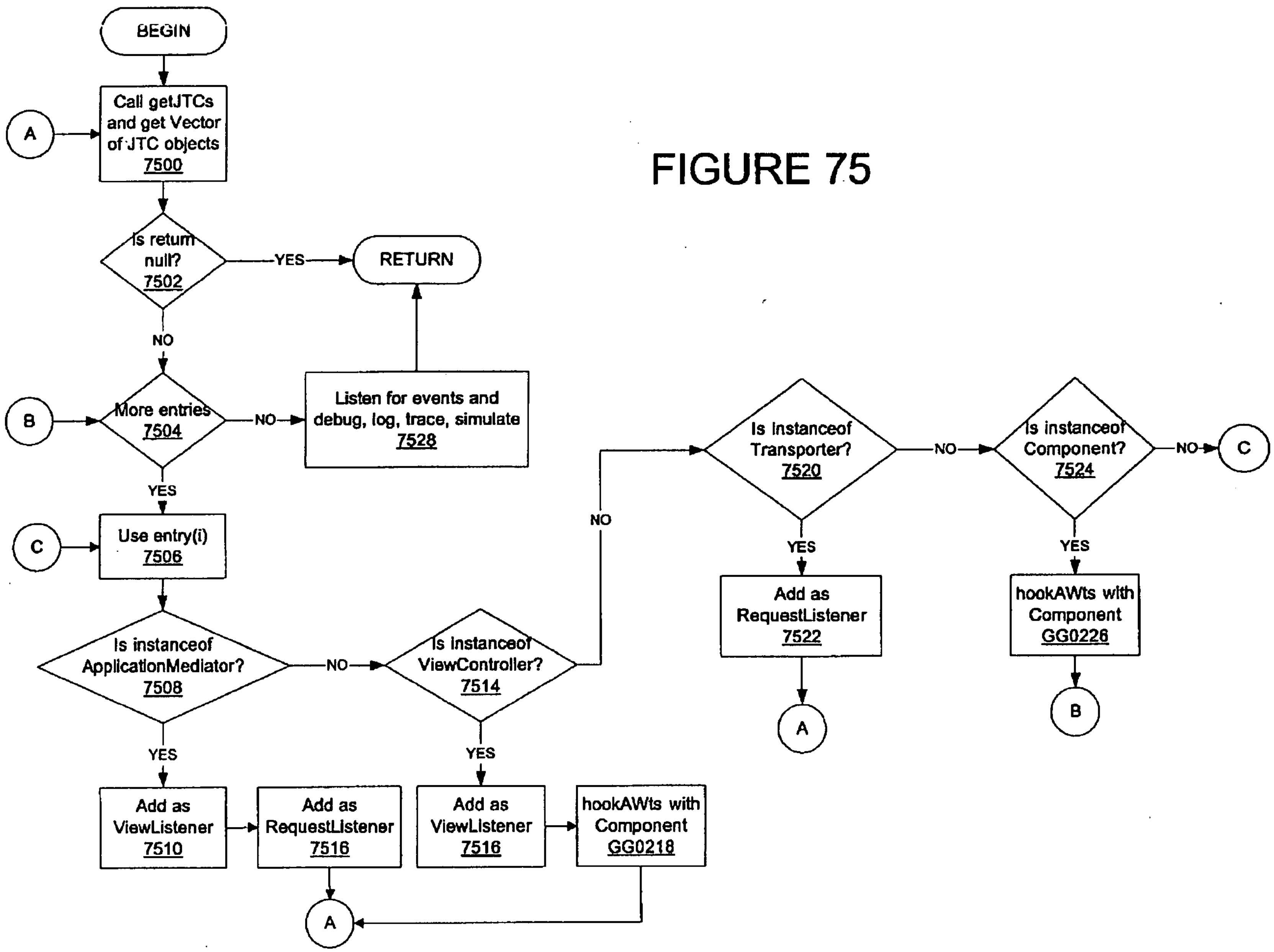
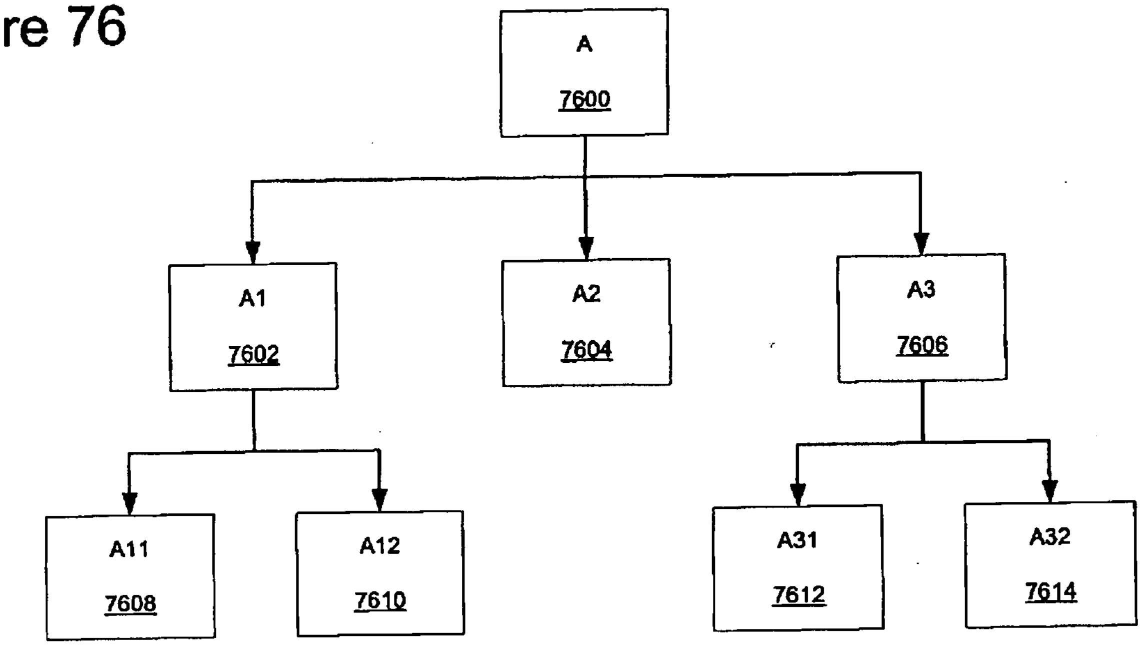

BEGIN
A
Call getJTCs and get Vector of JTC objects 7500
Is return null? 7502
YES
RETURN
NO
B
More entries 7504
NO
Listen for events and debug, log, trace, simulate 7528
YES
C
Use entry(i) 7506
Is instanceof ApplicationMediator? 7508
NO
Is instanceof ViewController? 7514
NO
Is Instanceof Transporter? 7520
NO
Is instanceof Component? 7524
NO
C
YES
Add as ViewListener 7510
Add as RequestListener 7516
A
YES
Add as ViewListener 7516
hookAWts with Component GG0218
YES
Add as RequestListener 7522
A
YES
hookAWts with Component GG0226
B

FIGURE 75

Figure 76


A
7600
A1
7602
A2
7604
A3
7606
A11
7608
A12
7610
A31
7612
A32
7614

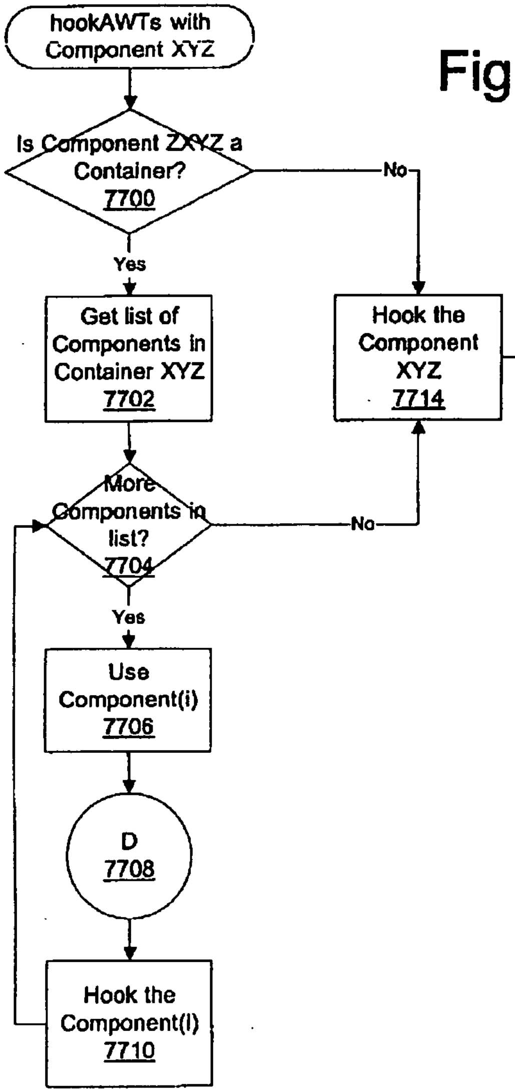


Figure 77

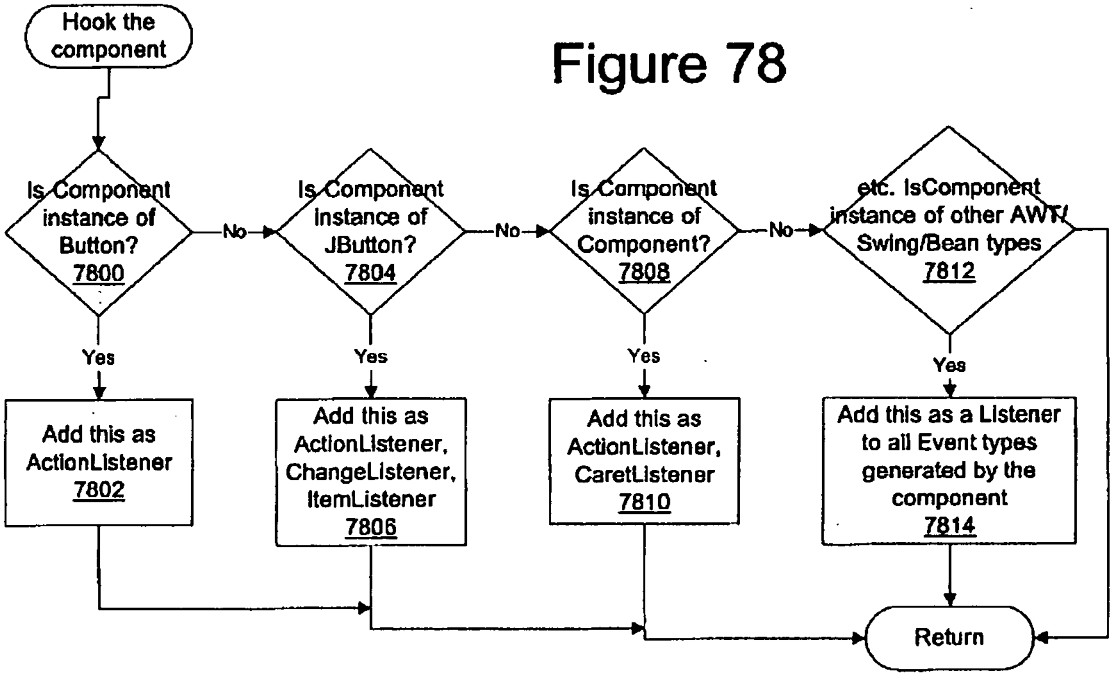

Hook the component
Is Component instance of Button?
7800
No
Is Component instance of JButton?
7804
No
Is Component instance of Component?
7808
No
etc. IsComponent instance of other AWT/ Swing/Bean types
7812
Yes
Add this as ActionListener
7802
Yes
Add this as ActionListener, ChangeListener, ItemListener
7806
Yes
Add this as ActionListener, CaretListener
7810
Yes
Add this as a Listener to all Event types generated by the component
7814
Return

Figure 78

Data Objects

√ The ApplicationMediatorImpl will forward the refresh (default)

FIGURE 79

```
for each: ApplicationMediator -> refresh(data)
for each: ViewController -> refresh(data);
```

√ The ViewController will update the GUI

FIGURE 80

```
public void refresh(Object data) {
 //this example uses a keyValue pair data model

    if (data == null) return;
    else refresh((KeyValue) data);
}

public void refresh (KeyValue data) {
    nameField.setText(data.get("CustomerName"));
    idField.setText(data.get("CustomerId"));
    repaint(); //if necessary
}
```

Data Objects

√ How can we add a new data model (i.e. real objects)?

FIGURE 81

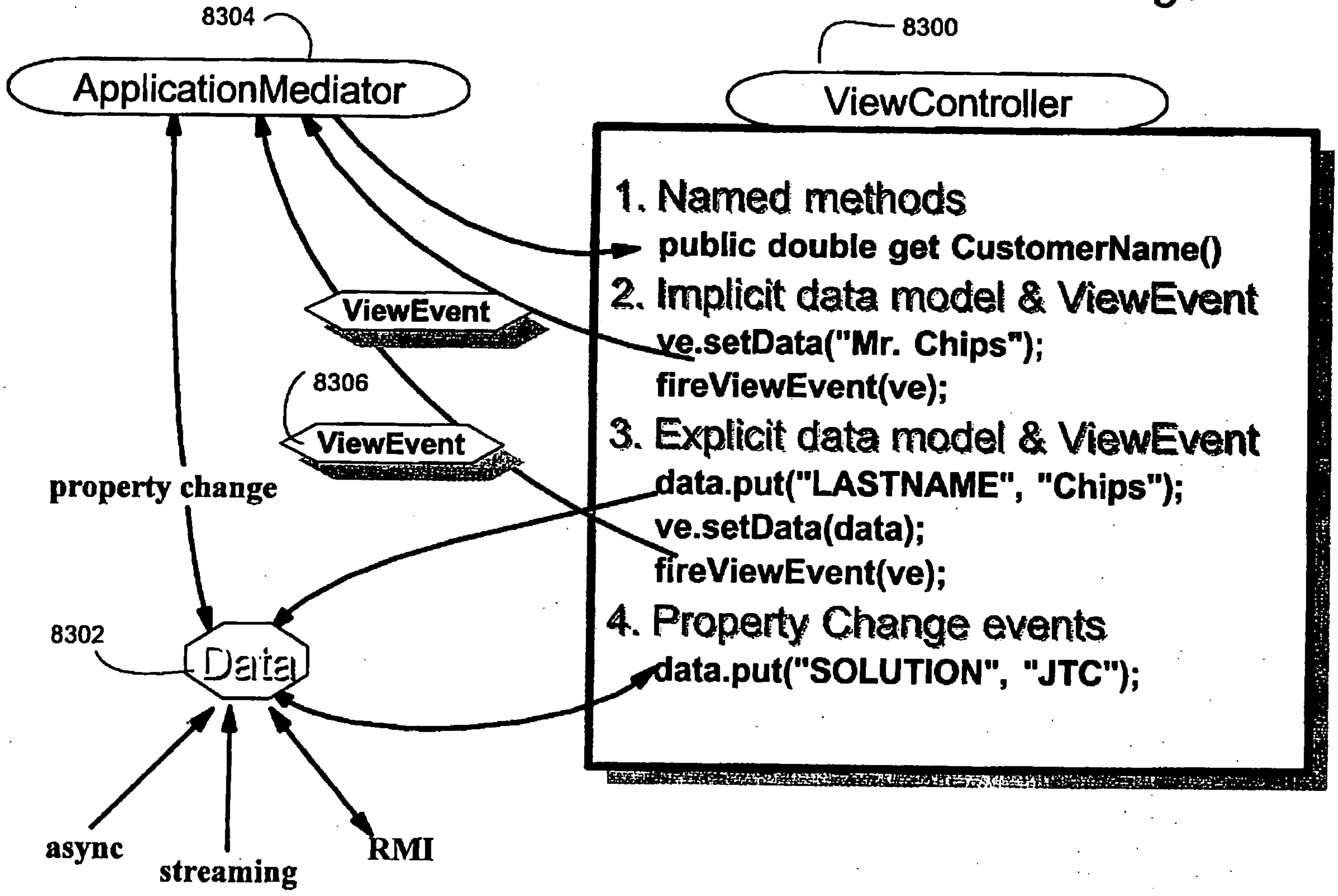
```
public void refresh(Object data) {
    if (data == null) return;
    else if (data instanceof Vector) {
        refresh((Vector) data);
    }
    else if (data instanceof KeyValue) {
            refresh((KeyValue) data);
        }
}
```

FIGURE 82

```
public void refresh(Vector data) {
    //I know what they are
    Customer c = (Customer) data.elementAt(0);
    ID id =  (ID) data.elementAt(1);
    nameField.setText(c.getName());
    idField.setText(id.toString());
    repaint(); //if necessary
}
```

More on data

Figure 83



1. Named methods
public double get CustomerName()
2. Implicit data model & ViewEvent
ve.setData("Mr. Chips");
fireViewEvent(ve);
3. Explicit data model & ViewEvent
data.put("LASTNAME", "Chips");
ve.setData(data);
fireViewEvent(ve);
4. Property Change events
data.put("SOLUTION", "JTC");

Figure 84

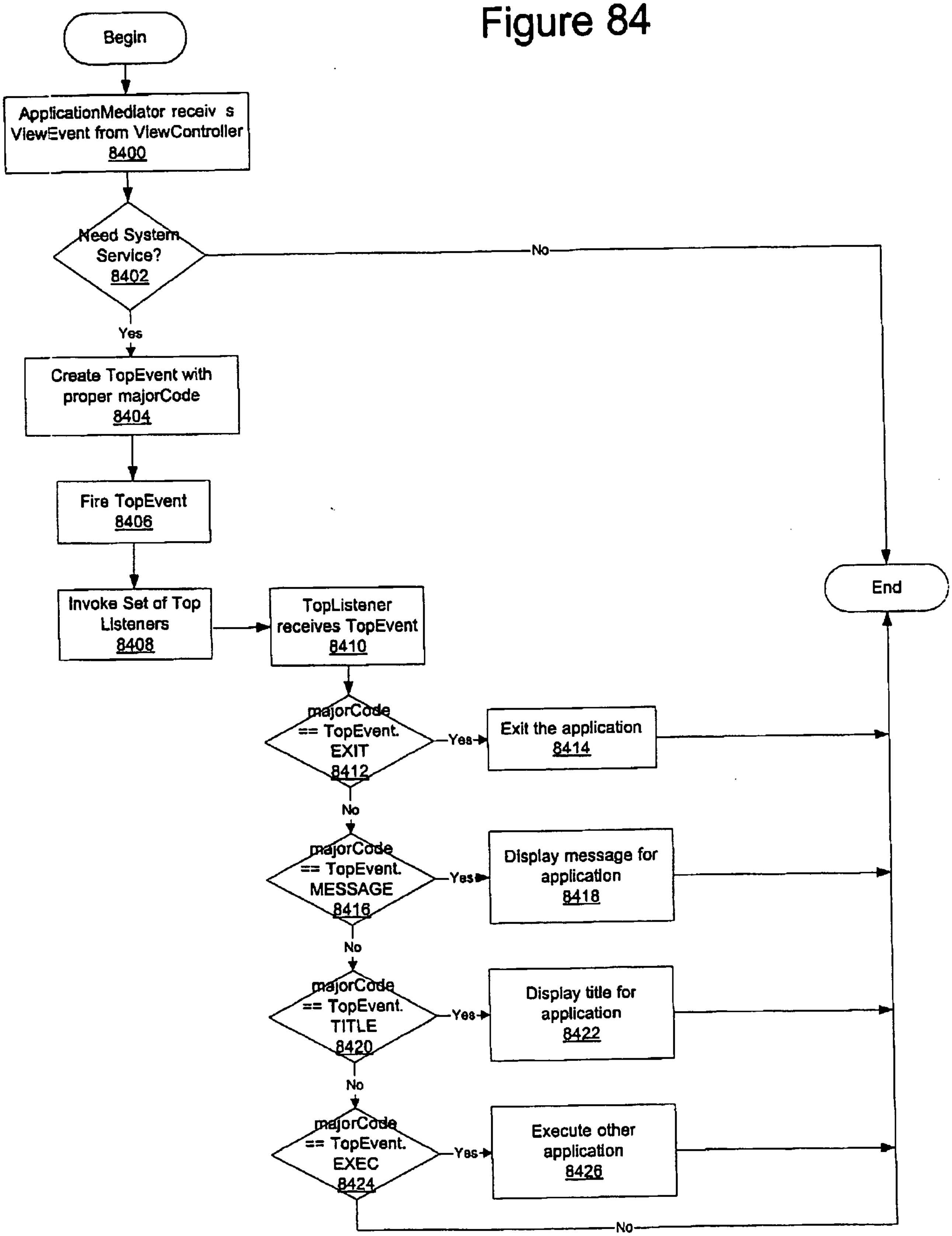

Begin
ApplicationMediator receiv s ViewEvent from ViewController 8400
Need System Service? 8402
No
Yes
Create TopEvent with proper majorCode 8404
Fire TopEvent 8406
Invoke Set of Top Listeners 8408
TopListener receives TopEvent 8410
majorCode == TopEvent. EXIT 8412
Yes
Exit the application 8414
No
majorCode == TopEvent. MESSAGE 8416
Yes
Display message for application 8418
No
majorCode == TopEvent. TITLE 8420
Yes
Display title for application 8422
No
majorCode == TopEvent. EXEC 8424
Yes
Execute other application 8426
No
End

Figure 85

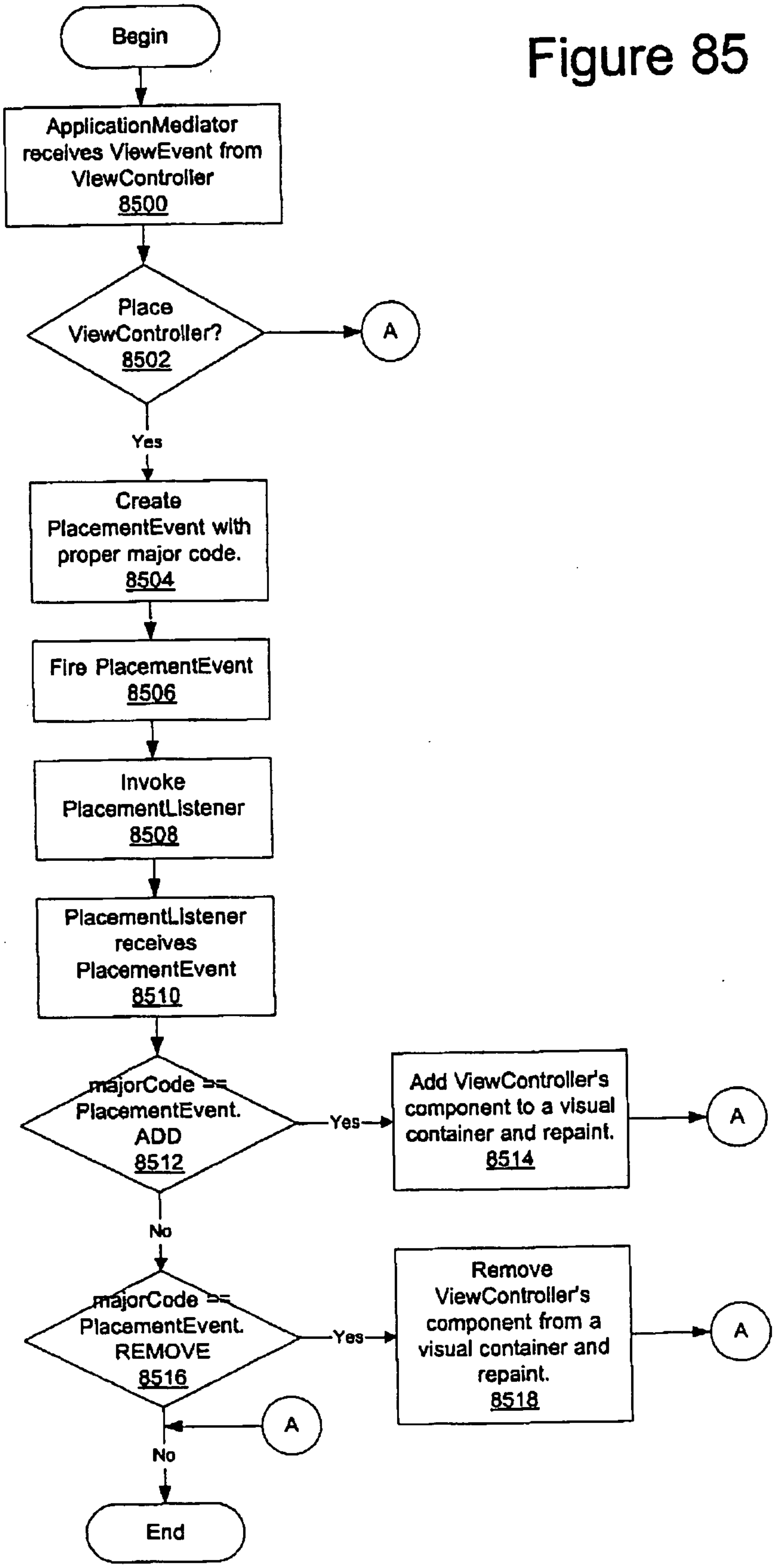
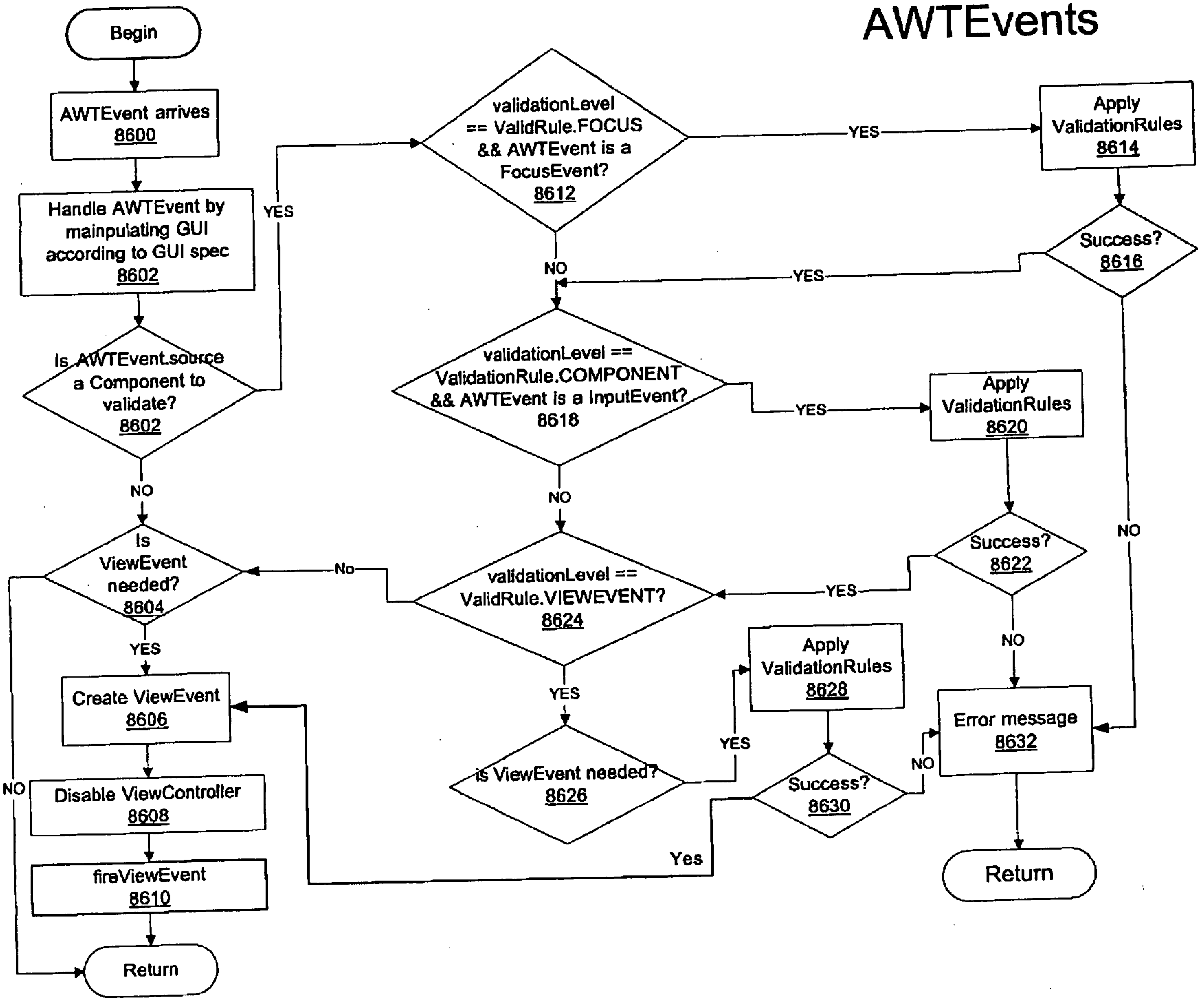

Begin
ApplicationMediator receives ViewEvent from ViewController 8500
Place ViewController? 8502
A
Yes
Create PlacementEvent with proper major code. 8504
Fire PlacementEvent 8506
Invoke PlacementListener 8508
PlacementListener receives PlacementEvent 8510
majorCode == PlacementEvent. ADD 8512
Yes
Add ViewController's component to a visual container and repaint. 8514
A
No
majorCode == PlacementEvent. REMOVE 8516
Yes
Remove ViewController's component from a visual container and repaint. 8518
A
A
No
End

FIGURE 86

Handling AWTEvents



Begin
AWTEvent arrives 8600
Handle AWTEvent by mainpulating GUI according to GUI spec 8602
Is AWTEvent.source a Component to validate? 8602
YES
NO
validationLevel == ValidRule.FOCUS && AWTEvent is a FocusEvent? 8612
YES
Apply ValidationRules 8614
Success? 8616
YES
NO
NO
validationLevel == ValidationRule.COMPONENT && AWTEvent is a InputEvent? 8618
YES
Apply ValidationRules 8620
Success? 8622
YES
NO
NO
validationLevel == ValidRule.VIEWEVENT? 8624
No
YES
Is ViewEvent needed? 8626
YES
Apply ValidationRules 8628
Success? 8630
NO
Yes
Error message 8632
Return
Is ViewEvent needed? 8604
YES
NO
Create ViewEvent 8606
Disable ViewController 8608
fireViewEvent 8610
Return

FIGURE 87

Applying ValidationRules

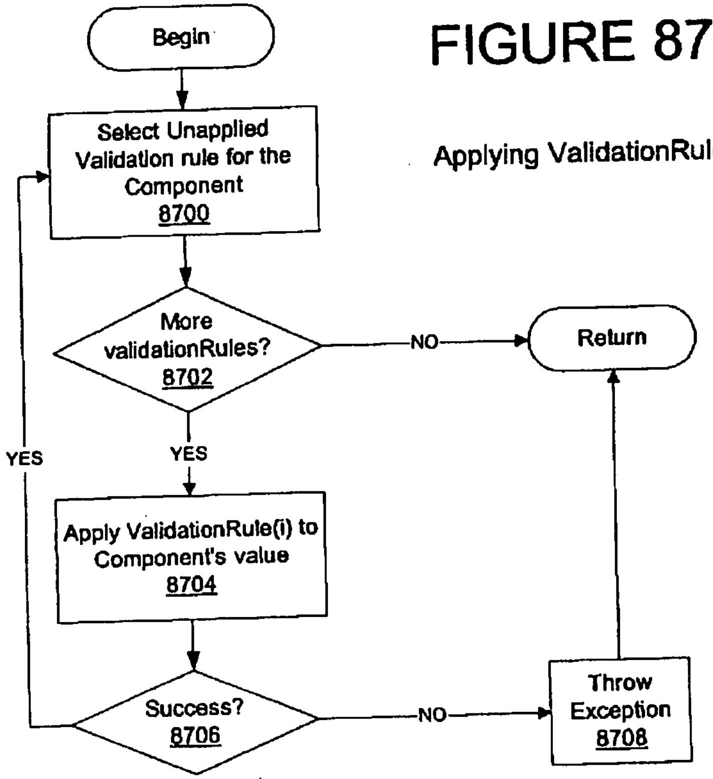

Begin
Select Unapplied Validation rule for the Component 8700
More validationRules? 8702
NO
Return
YES
Apply ValidationRule(i) to Component's value 8704
YES
Success? 8706
NO
Throw Exception 8708

FIGURE 88

fireViewEvent

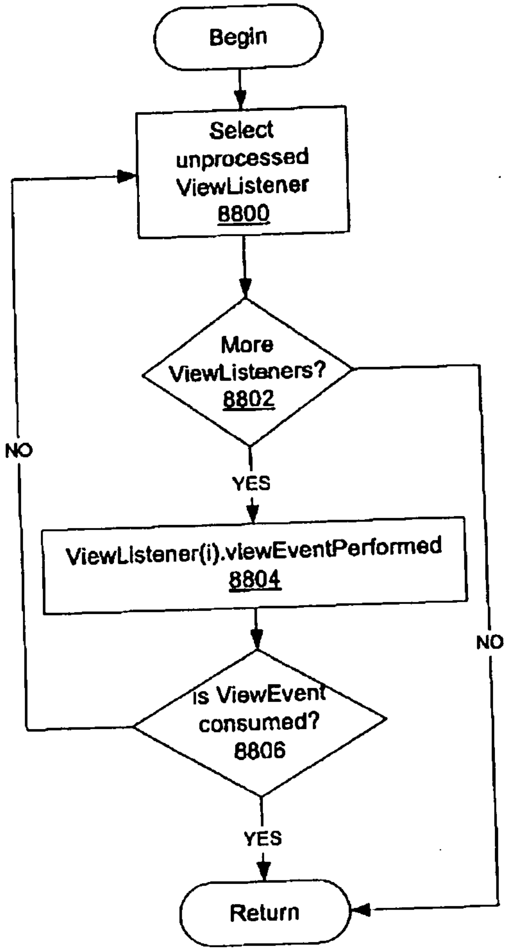

Begin
Select unprocessed ViewListener 8800
More ViewListeners? 8802
YES
ViewListener(i).viewEventPerformed 8804
NO
Is ViewEvent consumed? 8806
NO
YES
Return

Figure 89

Application Manager

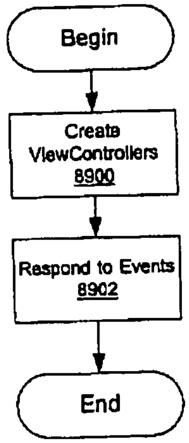


Figure 90

Application Manag r

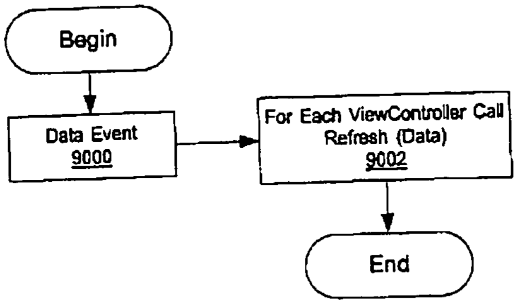


Figure 91

Application Manager

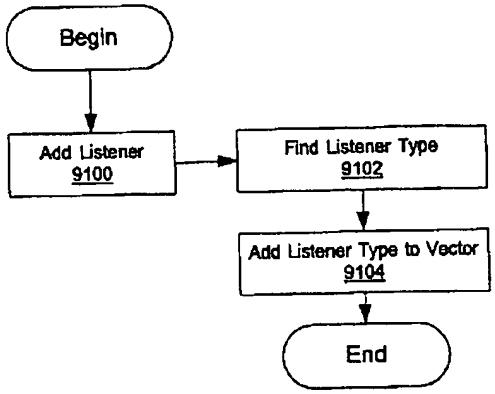


Figure 92

Application Manager

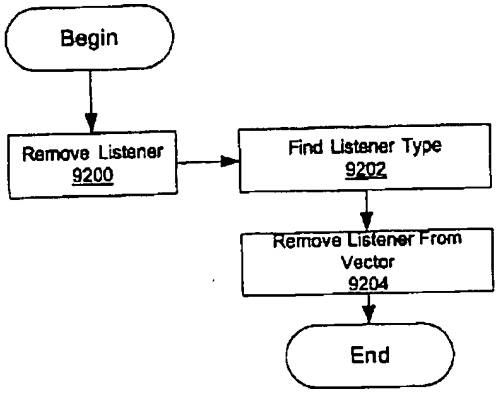


Figure 93

Application Manager

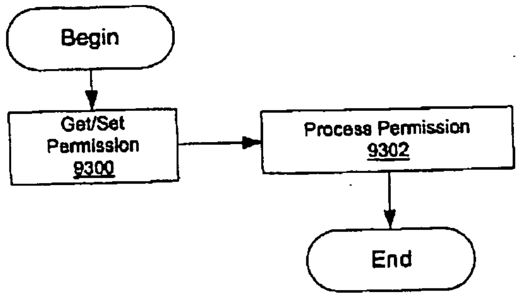
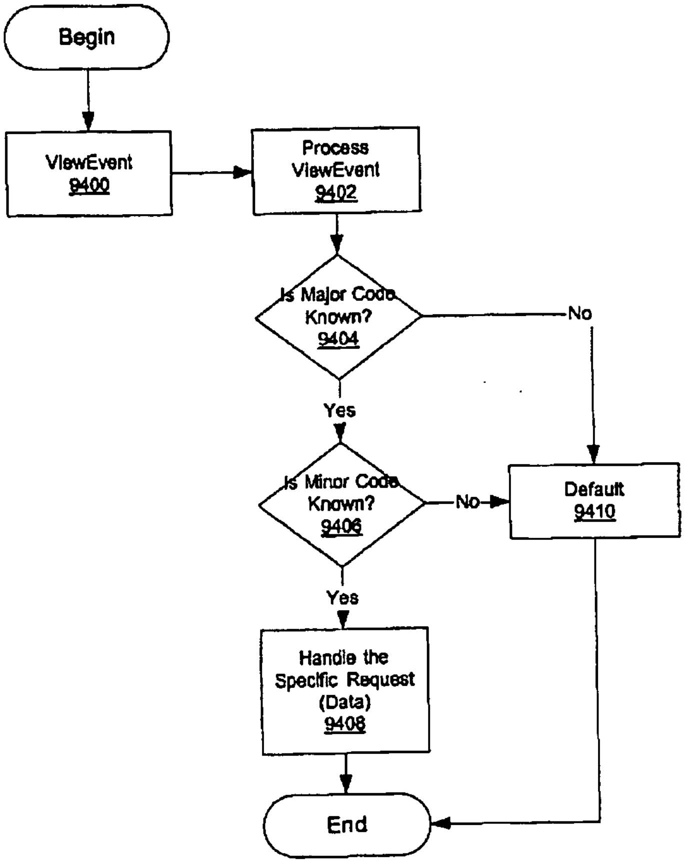


Figure 94

Application Manager



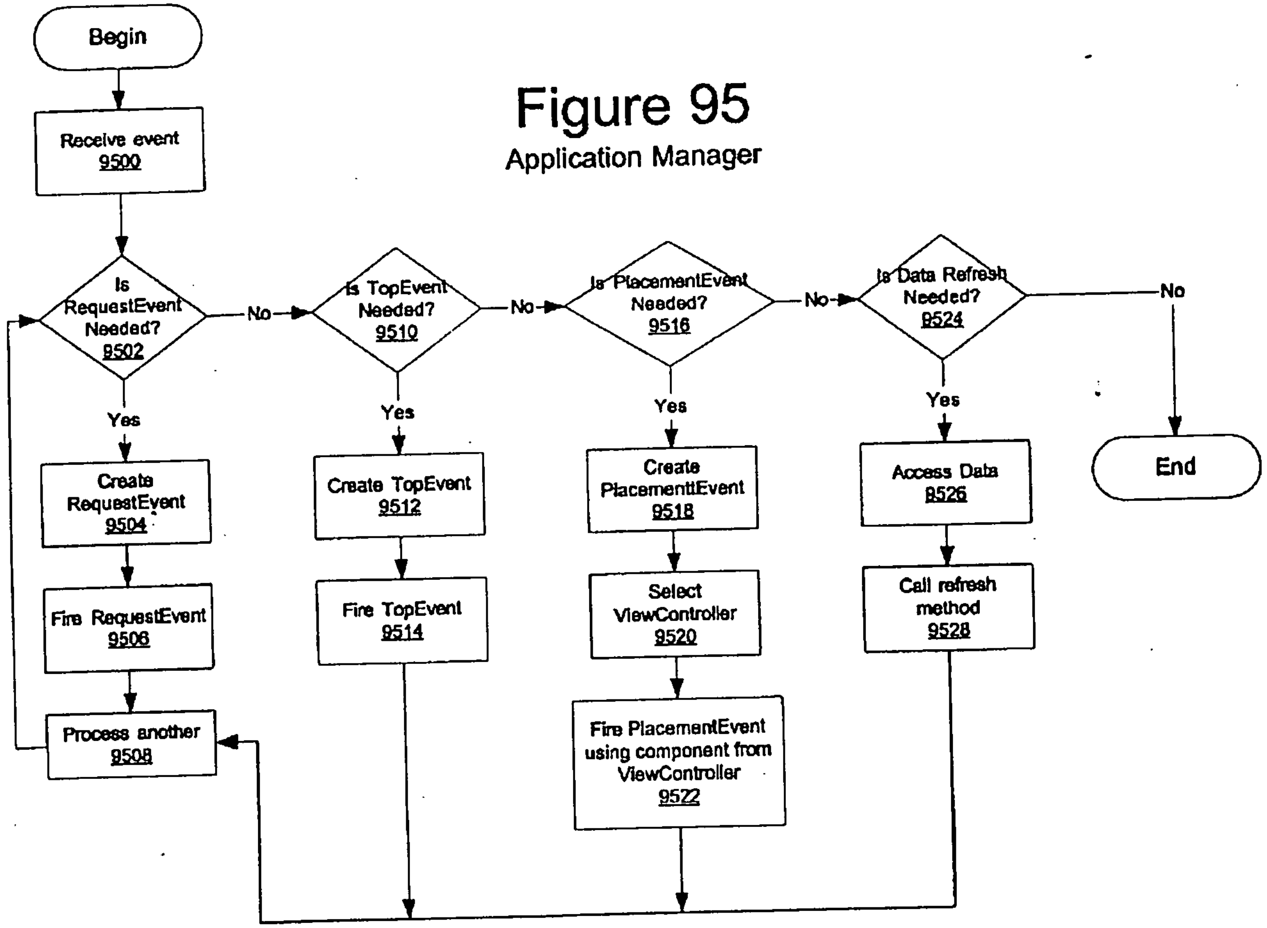


Figure 95
Application Manager

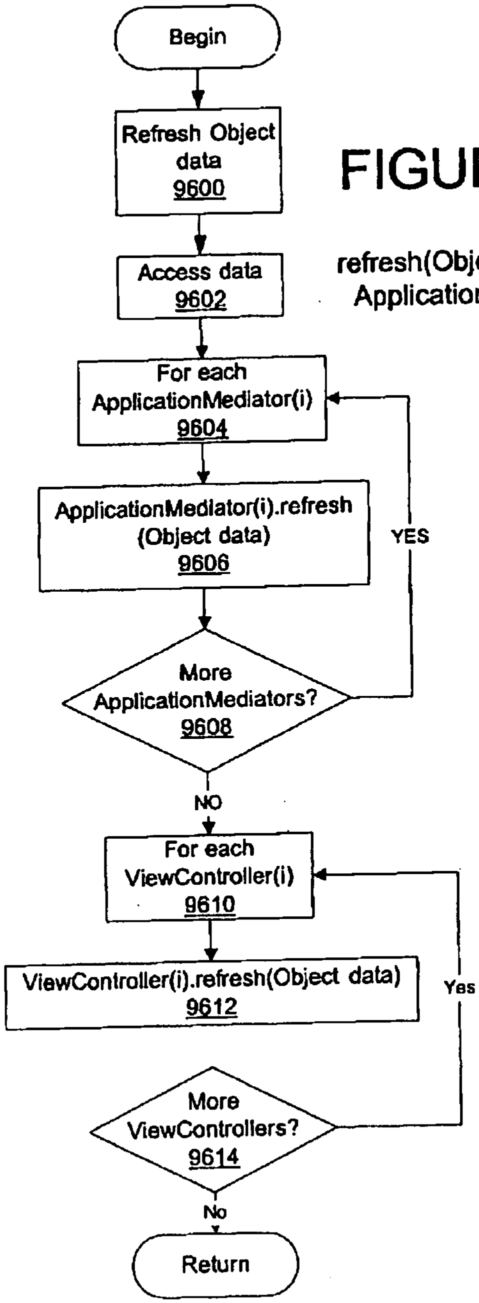


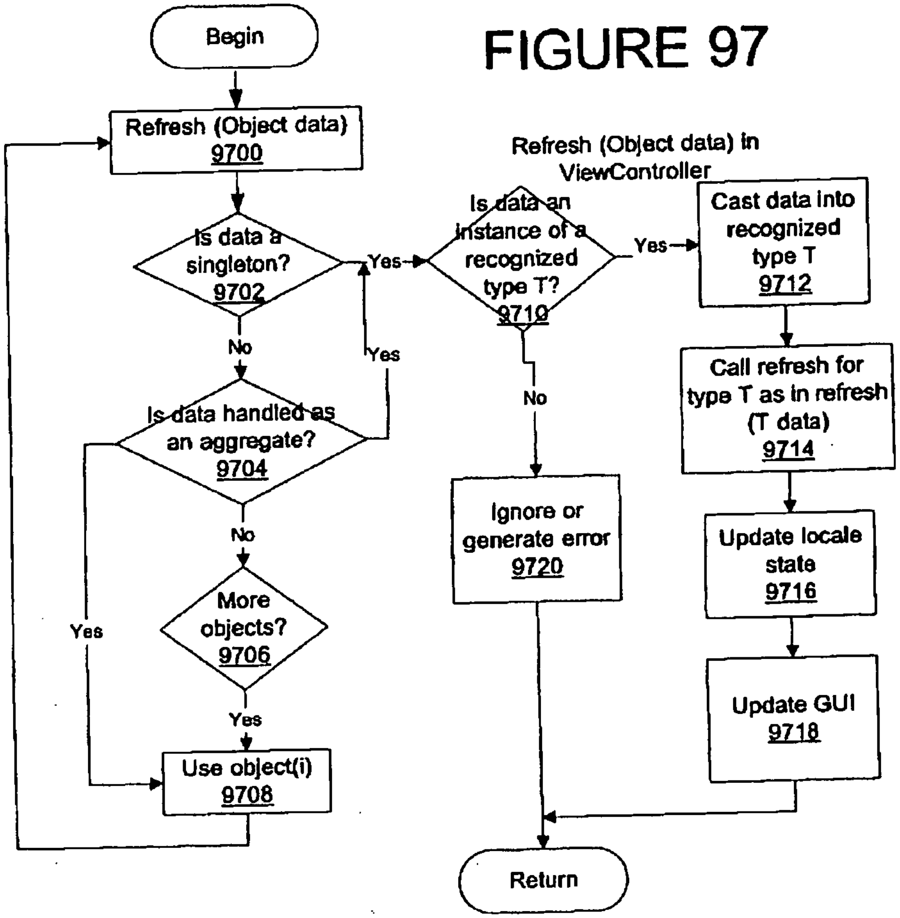
FIGURE 96

refresh(Object data) in ApplicationMediator

Begin
Refresh Object data 9600
Access data 9602
For each ApplicationMediator(i) 9604
ApplicationMediator(i).refresh (Object data) 9606
More ApplicationMediators? 9608
YES
NO
For each ViewController(i) 9610
ViewController(i).refresh(Object data) 9612
More ViewControllers? 9614
Yes
No
Return

FIGURE 97

Refresh (Object data) in ViewController



Begin
Refresh (Object data) 9700
Is data a singleton? 9702
Yes
No
Is data handled as an aggregate? 9704
Yes
Yes
No
More objects? 9706
Yes
Use object(i) 9708
Is data an instance of a recognized type T? 9710
Yes
No
Cast data into recognized type T 9712
Call refresh for type T as in refresh (T data) 9714
Update locale state 9716
Update GUI 9718
Ignore or generate error 9720
Return

FIGURE 98

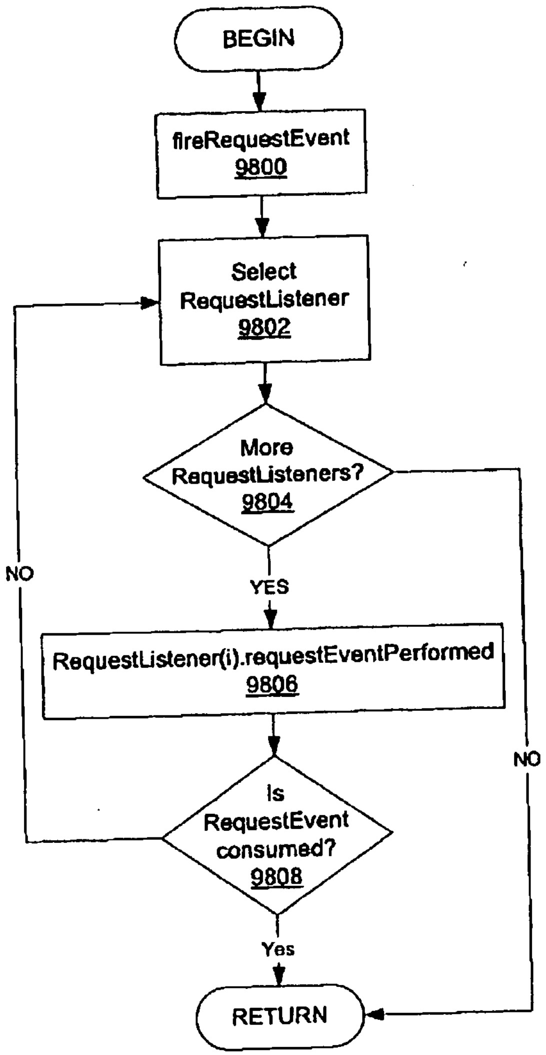
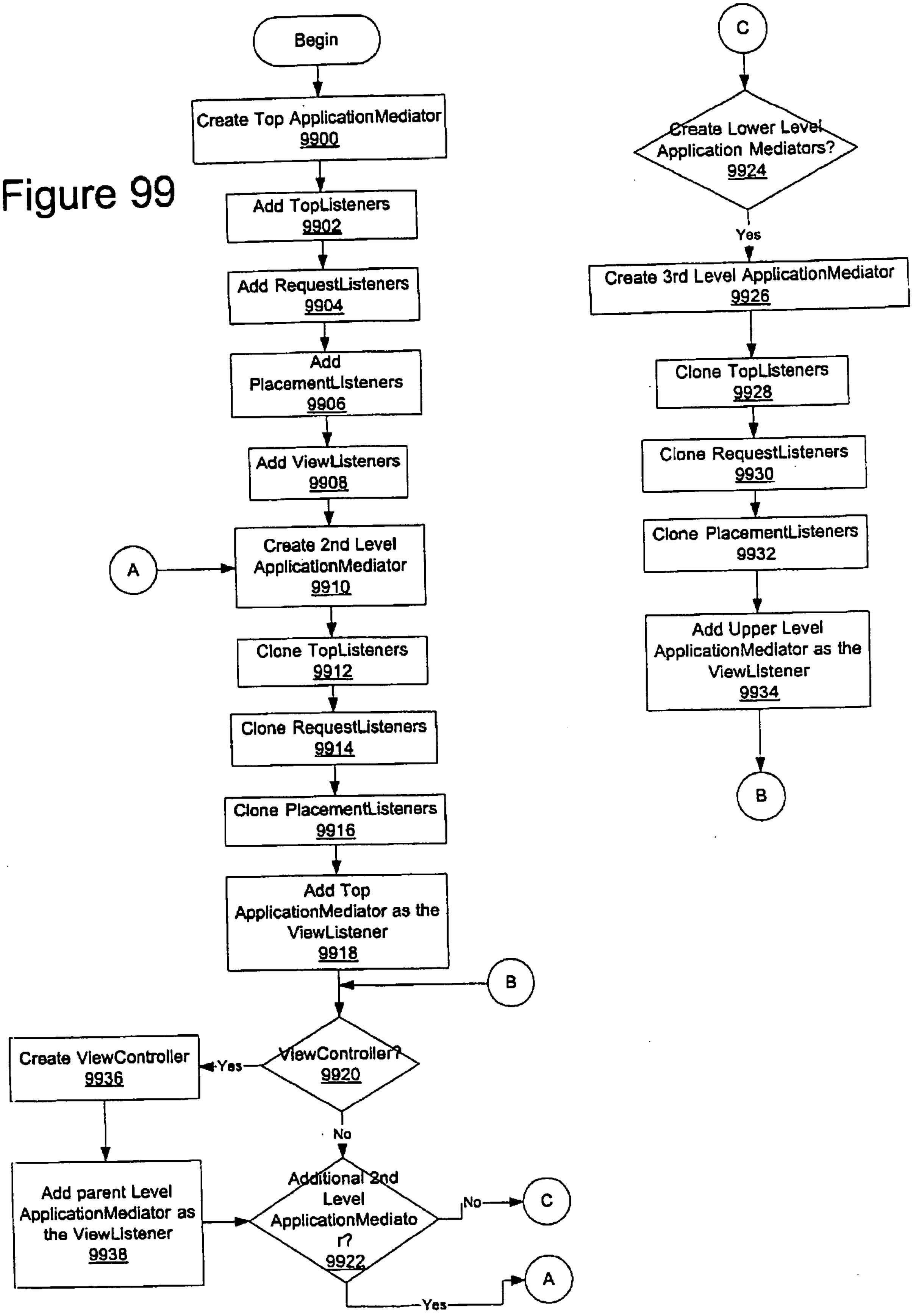

BEGIN
fireRequestEvent
9800
Select
RequestListener
9802
More
RequestListeners?
9804
YES
NO
RequestListener(i).requestEventPerformed
9806
Is
RequestEvent
consumed?
9808
Yes
NO
RETURN

Figure 99


Begin
Create Top ApplicationMediator 9900
Add TopListeners 9902
Add RequestListeners 9904
Add PlacementListeners 9906
Add ViewListeners 9908
A
Create 2nd Level ApplicationMediator 9910
Clone TopListeners 9912
Clone RequestListeners 9914
Clone PlacementListeners 9916
Add Top ApplicationMediator as the ViewListener 9918
B
ViewController? 9920
Yes
Create ViewController 9936
Add parent Level ApplicationMediator as the ViewListener 9938
No
Additional 2nd Level ApplicationMediato r? 9922
No
C
Yes
A
C
Create Lower Level Application Mediators? 9924
Yes
Create 3rd Level ApplicationMediator 9926
Clone TopListeners 9928
Clone RequestListeners 9930
Clone PlacementListeners 9932
Add Upper Level ApplicationMediator as the ViewListener 9934
B

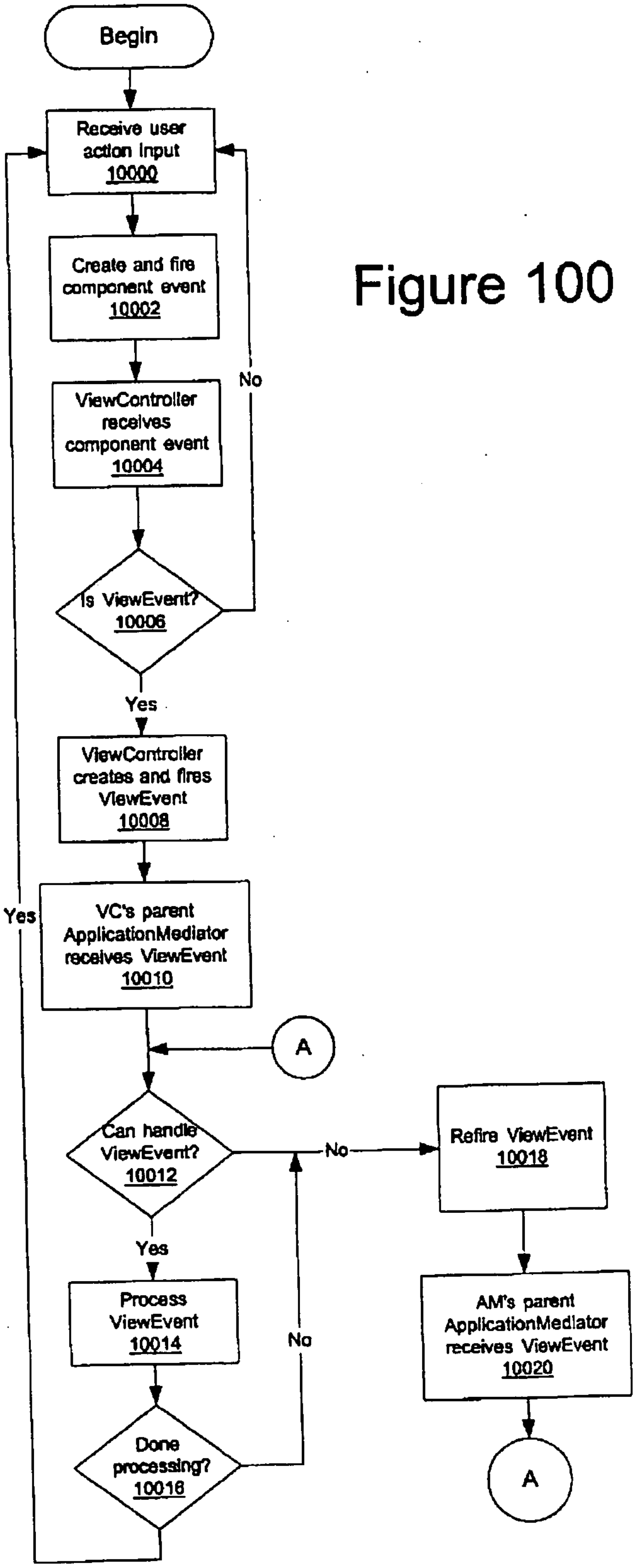

Begin
Receive user action input 10000
Create and fire component event 10002
ViewController receives component event 10004
Is ViewEvent? 10006
No
Yes
ViewController creates and fires ViewEvent 10008
VC's parent ApplicationMediator receives ViewEvent 10010
Yes
A
Can handle ViewEvent? 10012
No
Refire ViewEvent 10018
Yes
Process ViewEvent 10014
No
AM's parent ApplicationMediator receives ViewEvent 10020
Done processing? 10016
A

Figure 100

FIGURE 101

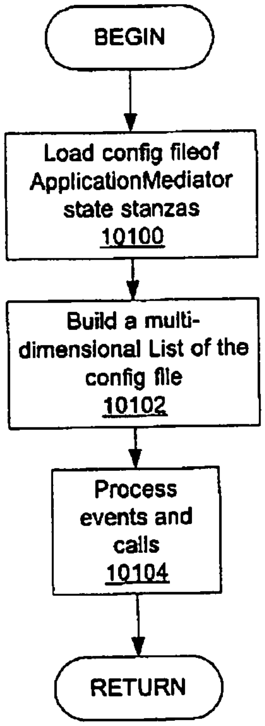

BEGIN
Load config fileof ApplicationMediator state stanzas 10100
Build a multi-dimensional List of the config file 10102
Process events and calls 10104
RETURN

FIGURE 102

Encoding ApplicationMediators

➡ S1: (VE.source==vc1 && VE.major==A && VE.minor==B) =>
(RE.major=C RE.minor=D RE.data=VE.data RE.fireS)

```
if event source is vc1 with A,B as major/minor then
    fire sync request with C,D major/minor and use data from event)
```

➡ S2: (VE.source==vc4 && VE.major==5) ==> (TE.major=3 TE.fire)

```
if event source is vc4 with 5 as major then
    fire top event with major 3
```

➡ S3: (Refresh) ==> (VC.i.refresh(Refresh.data))

```
if refresh(data) occurs, then refresh all view controllers with the
same data, but not the other application mediators
```

➡ S4: (VE.source==vcA) ==> (RE.major="set" RE.fireA) &&
(PE.major=PE.ADD PE.source=vcB PE.fire) && (VC.vcB.refresh(RE.data))

```
if event source is vcA, then fire async request, then fire placement
event, then refresh the newly placed view controller with the data
returned with the request
```

FIGURE 103

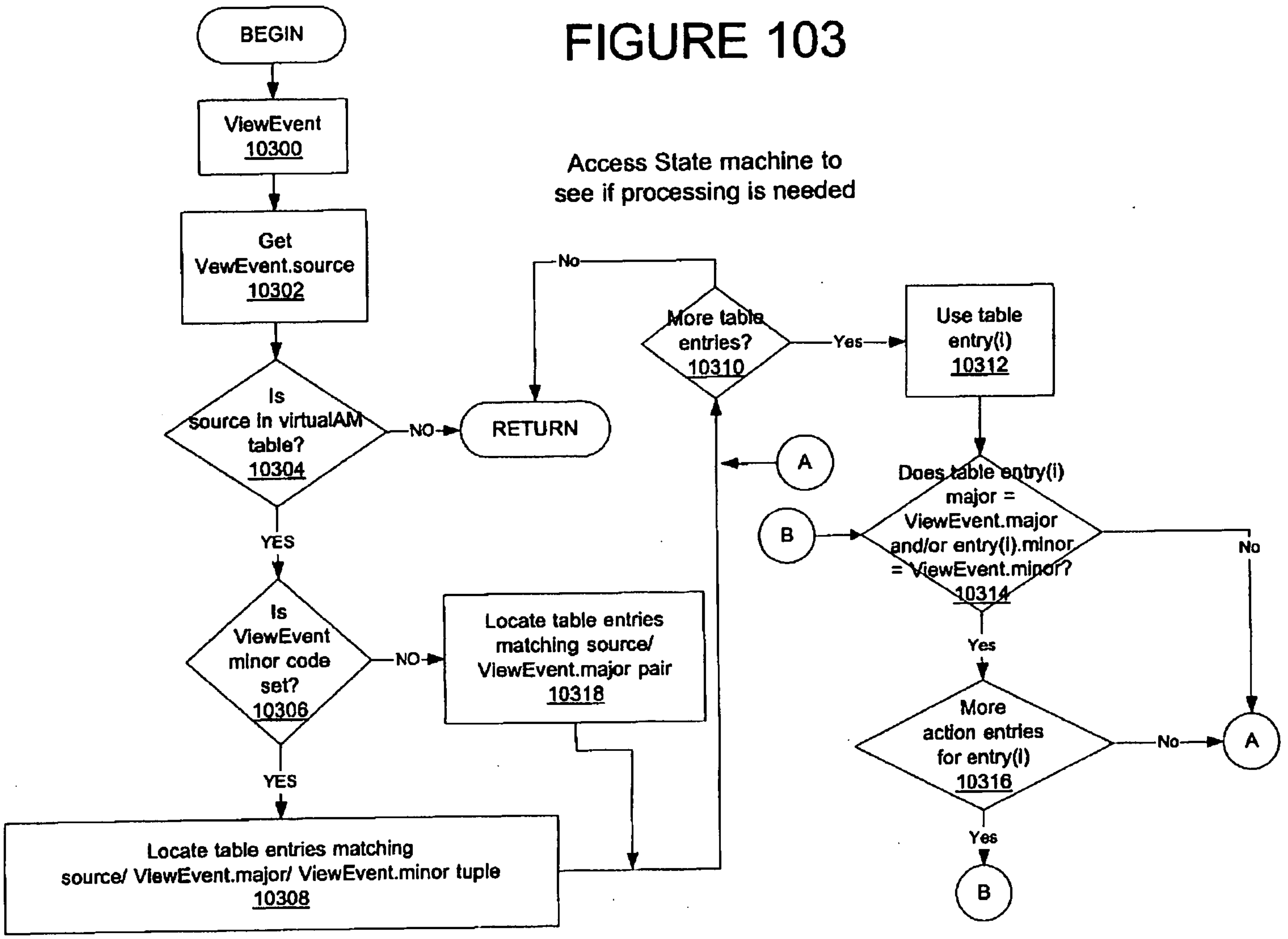

BEGIN
ViewEvent 10300
Get VewEvent.source 10302
Is source in virtualAM table? 10304
NO
RETURN
YES
Is ViewEvent minor code set? 10306
NO
Locate table entries matching source/ ViewEvent.major pair 10318
YES
Locate table entries matching source/ ViewEvent.major/ ViewEvent.minor tuple 10308
Access State machine to see if processing is needed
No
More table entries? 10310
Yes
Use table entry(I) 10312
A
B
Does table entry(I) major = ViewEvent.major and/or entry(I).minor = ViewEvent.minor? 10314
No
Yes
More action entries for entry(I) 10316
No
A
Yes
B

FIGURE 104

Process actions for each ViewEvent

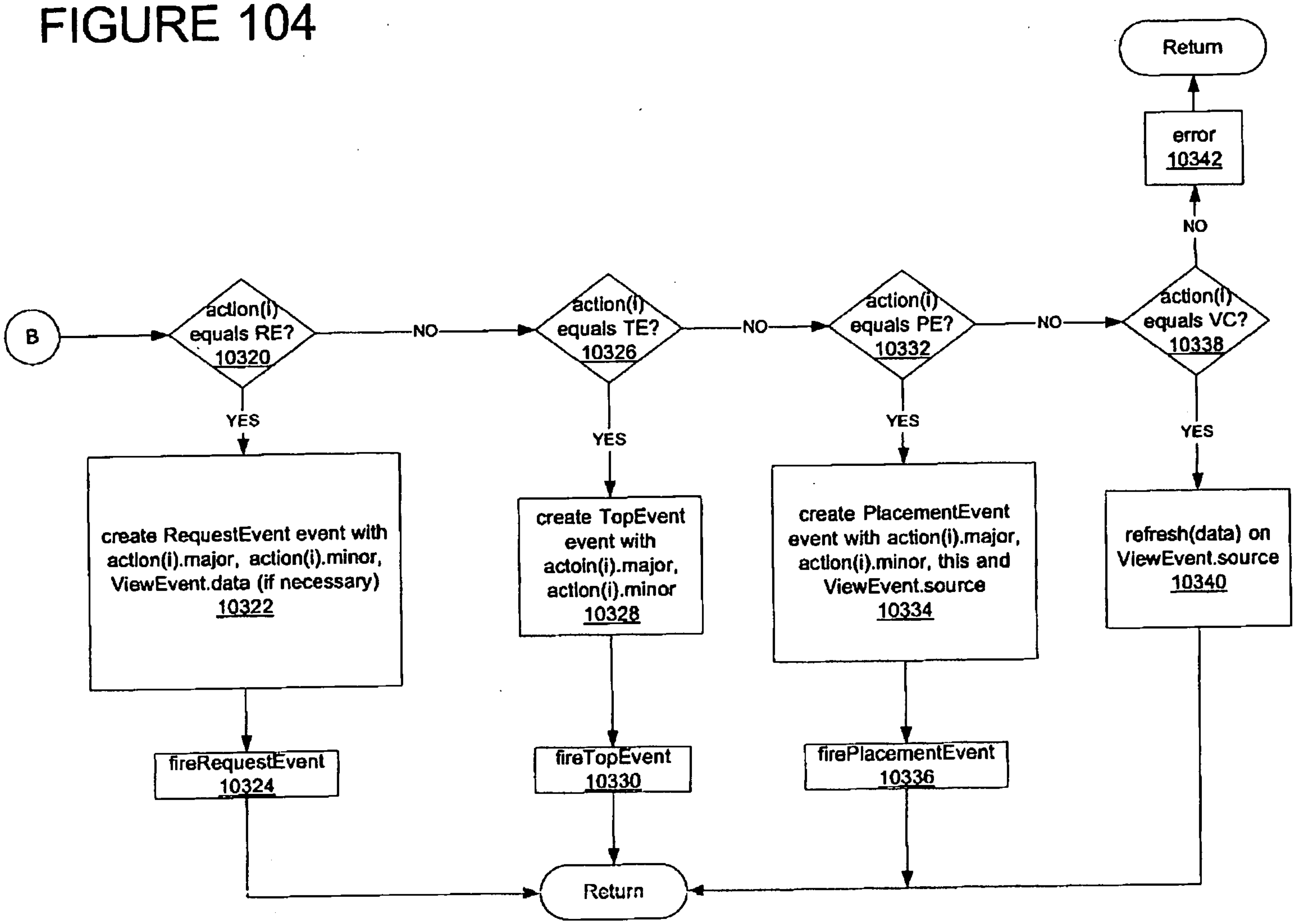

B
action(i) equals RE? 10320
NO
action(i) equals TE? 10326
NO
action(i) equals PE? 10332
NO
action(i) equals VC? 10338
NO
error 10342
Return
YES
create RequestEvent event with action(i).major, action(i).minor, ViewEvent.data (if necessary) 10322
fireRequestEvent 10324
YES
create TopEvent event with actoin(i).major, action(i).minor 10328
fireTopEvent 10330
YES
create PlacementEvent event with action(i).major, action(i).minor, this and ViewEvent.source 10334
firePlacementEvent 10336
YES
refresh(data) on ViewEvent.source 10340
Return

Figure 105

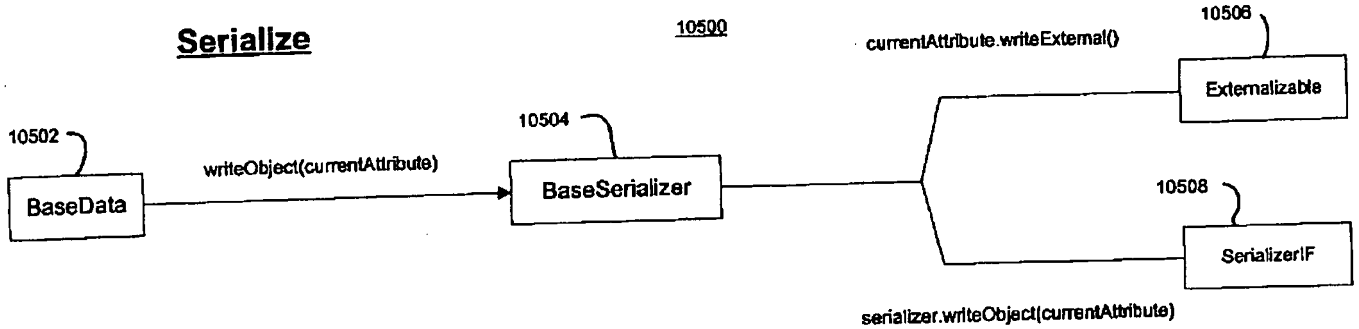

Serialize
10500
10502
BaseData
writeObject(currentAttribute)
10504
BaseSerializer
currentAttribute.writeExternal()
10506
Externalizable
10508
SerializerIF
serializer.writeObject(currentAttribute)

Figure 106

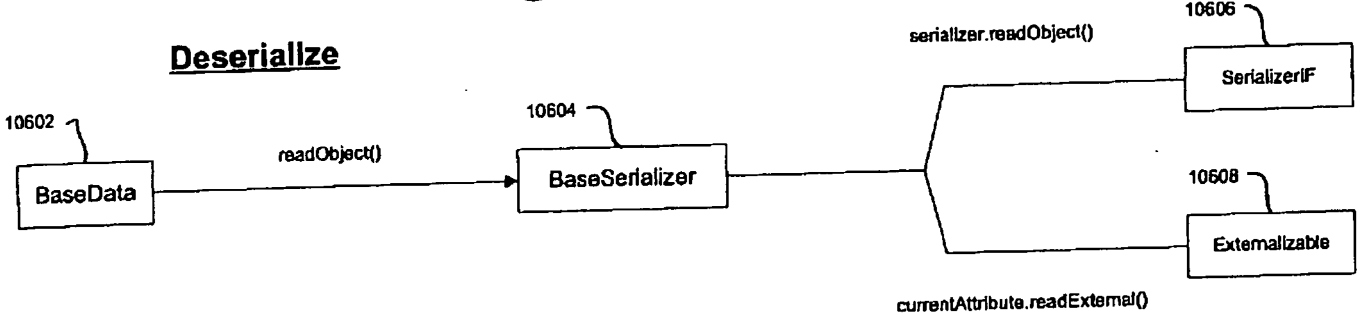


Figure 107

10700
pg 1

```
package com.ibm.jtcx.serialization;

import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
/**
 * Default type comment.
 *
 * <P>INVARIANT:
 */
public class BaseData implements Externalizable {
	private Object[] data = null;
/**
 * BaseData constructor comment.
 */
public BaseData() {
	this(0);
}
/**
 * BaseData constructor comment.
 * @param dataArray java.lang.Object[]
 */
public BaseData(int count) {
	super();

	setData(new Object[count]);
}
/**
 * Default method comment.
 *
 * <P>PRE:
 * <P>POST:
 *
 * @return Parameter not modified
 * @return java.lang.Object[]
 */
public final Object[] getData() {
	return data;
}
```

Figure 107

10700
pg 2

```
/**
 * Default method comment.
 *
 * <P>PRE:
 * <P>POST:
 *
 * @return Parameter not modified
 * @return java.lang.Object
 * @param index int
 */
public final Object getData(int index) {
	Object retVal = null;

	if ((data != null) && (index < data.length)) {
		retVal = data[index];
	}

	return retVal;
}
/**
 * Default method comment.
 *
 * <P>PRE:
 * <P>POST:
 *
 * @return Parameter not modified
 * @param in ObjectInput
 */
public void readExternal(ObjectInput in)
	throws ClassNotFoundException, IOException {

	setData((Object[])in.readObject());
}
/**
 * Default method comment.
 *
 * <P>PRE:
 * <P>POST:
 *
 * @return Parameter not modified
 * @param dataArray java.lang.Object[]
 */
public final void setData(Object[] dataArray) {
	data = dataArray;
}
```

Figure 107

10700
pg 3

```
/**
 * Default method comment.
 *
 * <P>PRE:
 * <P>POST:
 *
 * @return Parameter not modified
 * @param index int
 * @param dataElement java.lang.Object
 */
public final void setData(int index, Object dataElement) {
        if ((data != null) && (index < data.length)) {
                data[index] = dataElement;
        }
}
/**
 * Default method comment.
 *
 * <P>PRE:
 * <P>POST:
 *
 * @return Parameter not modified
 * @param out ObjectOutput
 */
public void writeExternal(ObjectOutput out) throws IOException {
        out.writeObject(getData());
}
}
```

Figure 108

10800
pg 1

```
package com.ibm.jtcx.serialization;

import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.Date;
import java.util.Enumeration;
import java.util.GregorianCalendar;
import java.util.Hashtable;
import java.util.SimpleTimeZone;
import java.util.TimeZone;
import java.util.Vector;
/**
 * Base class of data objects that require small serialization. The
 * attributes of the data object are stored in an array and the elements
 * of the array are written individually.
 *
 * <P>INVARIANT:
 */
public class BaseDataS extends BaseData implements Externalizable {
/**
 * Default constructor.
 */
public BaseDataS() {
        super();
}
/**
 * Creates a new <code>BaseDataS</code> object with a data array of
 * size <code>count</code>.
 *
 * @param count the size of the data array containing the attributes
 */
public BaseDataS(int count) {
        super(count);
}
/**
 * Reads the array of data elements from the stream.  The responsibility
 * of reading the individual element is left to the
 * <code>BaseSerializer</code> via <code>readObject()<code>.
 *
 * @param in the input stream that contains the serialized object
 * @exception ClassNotFoundException thrown if
 * <code>BaseSerializer</code> fails to read the object from the stream
 * @exception IOException thrown if
 * <code>BaseSerializer</code> fails to read the object from the stream
 * @see BaseSerializer#readObject
 */
public void readExternal(ObjectInput in)
        throws ClassNotFoundException, IOException {

        int size = in.readShort();

        if (size == -1) {
                setData(null);
        } else {
                Object[] array = new Object[size];

                for (int i = 0; i < size; i++) {
                        array[i] = BaseSerializer.getInstance().readObject(in);
                }

                setData(array);
        }
}
/**
 * Writes the array of data elements.  The responsibility of writing the
 * data elements is left to <code>BaseSerializer</code> via
 * <code>writeObject()</code>.
 *
 * @param out the output stream to which the data elements will be
 * written
 */
public void writeExternal(ObjectOutput out) throws IOException {
        Object[] array = getData();

        if (array == null) {
                out.writeShort(-1);
        } else {
                out.writeShort(array.length);

                for (int i = 0; i < array.length; i++) {
                        BaseSerializer.getInstance().writeObject(out, array[i]);
                }
        }
}
}
```

Figure 108

10800
pg 2

Figure 109

10900
pg 1

```
package com.ibm.jtcx.serialization;

import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
/**
 * The interface for those classes that serialize objects to and from
 * a stream. The object that implements this interface should write
 * just the object's attributes, not any other descriptive information
 * about the object. Typically, a <code>SerializerIF</code> knows how
 * to serialize a specific class.
 */
public interface SerializerIF {
/**
 * Reads an object from the stream.
 *
 * @return The object that was read.
 * @param in the input stream containing the object
 * @exception java.io.IOException thrown if the stream fails
 * @exception java.lang.ClassNotFoundException thrown if the stream
 * fails
 */
Object readObject(ObjectInput in) throws IOException, ClassNotFoundException;
/**
 * Writes the given object to the stream.
 *
 * @param out the output stream into which the object will be written
 * @param element the object that will be written to the stream
 * @exception java.io.IOException thrown if the stream fails
 */
void writeObject(ObjectOutput out, Object element) throws IOException;
}
```

Figure 110

11000
pg 1

```
package com.ibm.jtcx.serialization;

import java.io.*;
import java.math.BigInteger;
import java.math.BigDecimal;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.Hashtable;
import java.util.SimpleTimeZone;
import java.util.StringTokenizer;
import java.util.TimeZone;
import java.util.Vector;

/**
 * The <code>SerializerIF</code> that is used as the base level
 * serializer.  It contains three tables used to serialize objects:
 * <br><ul>
 *              <li> codeTable: the table containing the serialization code of
 *                      an object based on the name of the class
 *              <li> nameTable: the table containing the name of the class
 *                      based on the serialization code
 *              <li> serializationTable: the table containing the serializer of
 *                      of an object based on its serialization code
 * </ul>
 * <br><br>
 * <code>BaseSerializer</code> delegates the responsiblity of
 * serializing the objects to the <code>SerializerIF</code> associated
 * with that class or to the object itself if it implements
 * <code>Externalizable</code>.
 */
public class BaseSerializer implements SerializerIF {
        static private final int NULL_OBJECT = 0;
        static private final int OTHER = 0x00ff;

        static private final String HASHTABLE_SER = "ClassNameHash.ser";
        static private final String INI_FILE = "ClassNames.ini";

        static private Hashtable codeTable = null;
        static private Hashtable nameTable = null;
        static private Hashtable serializerTable = null;
        static private BaseSerializer instance = null;

        class BigDecimalSerializer implements SerializerIF {
                public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
```

Figure 110

11000
pg 2

```
            int scale = in.readShort();
            int size = in.readShort();
            byte[] bytes = new byte[size];
            in.readFully(bytes);

            BigInteger temp = new BigInteger(bytes);
            return new BigDecimal(temp, scale);
        }
        public void writeObject(ObjectOutput out, Object element) throws IOException {
            BigDecimal bigD = (BigDecimal)element;

            int scale = bigD.scale();
            bigD.setScale(0);
            byte[] bytes = bigD.toBigInteger().toByteArray();
            bigD.setScale(scale);

            out.writeShort(scale);
            out.writeShort(bytes.length);
            out.write(bytes);
        }
    }
    class BigIntegerSerializer implements SerializerIF {
        public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException
{

            int size = in.readShort();
            byte[] bytes = new byte[size];
            in.readFully(bytes);

            return new BigInteger(bytes);
        }
        public void writeObject(ObjectOutput out, Object element) throws IOException {
            byte[] bytes = ((BigInteger)element).toByteArray();

            out.writeShort(bytes.length);
            out.write(bytes);
        }
    }
    class BooleanSerializer implements SerializerIF {
        public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException
{

            int value = in.readByte();

            return (value == 1) ? Boolean.TRUE : Boolean.FALSE;
        }
        public void writeObject(ObjectOutput out, Object element) throws IOException {
            out.writeByte(((Boolean)element).booleanValue() ? 1 : 0);
        }
    }
```

Figure 110

11000
pg 3

```
class ByteSerializer implements SerializerIF {
            public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
                    byte value = in.readByte();

                    return new Byte(value);
            }
            public void writeObject(ObjectOutput out, Object element) throws IOException {
                    out.writeByte(((Byte)element).byteValue());
            }
    }
    class CharacterSerializer implements SerializerIF {
            public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
                    char value = in.readChar();

                    return new Character(value);
            }
            public void writeObject(ObjectOutput out, Object element) throws IOException {
                    out.writeChar(((Character)element).charValue());
            }
    }
    class DateSerializer implements SerializerIF {
            public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
                    long value = in.readLong();

                    return new Date(value);
            }
            public void writeObject(ObjectOutput out, Object element) throws IOException {
                    out.writeLong(((Date)element).getTime());
            }
    }
    class DoubleSerializer implements SerializerIF {
            public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
                    double value = in.readDouble();

                    return new Double(value);
            }
            public void writeObject(ObjectOutput out, Object element) throws IOException {
                    out.writeDouble(((Double)element).doubleValue());
            }
    }
```

Figure 110

11000
pg 4

```
class FloatSerializer implements SerializerIF {
        public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
            float value = in.readFloat();

            return new Float(value);
        }
        public void writeObject(ObjectOutput out, Object element) throws IOException {
            out.writeFloat(((Float)element).floatValue());
        }
    }
    class GregorianCalendarSerializer implements SerializerIF {
        public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
            long time = in.readLong();
            Date date = new Date(time);
            SerializerIF serializer = BaseSerializer.getInstance();
            TimeZone tz = (TimeZone)serializer.readObject(in);

            GregorianCalendar gCalendar = new GregorianCalendar(tz);
            gCalendar.setTime(date);

            return gCalendar;
        }
        public void writeObject(ObjectOutput out, Object element) throws IOException {
            GregorianCalendar temp = (GregorianCalendar)element;

            Date date = temp.getTime();
            TimeZone tz = temp.getTimeZone();

            out.writeLong(date.getTime());
            SerializerIF serializer = BaseSerializer.getInstance();
            serializer.writeObject(out, tz);
        }
    }
    class IntegerSerializer implements SerializerIF {
        public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
            int value = in.readInt();

            return new Integer(value);
        }
        public void writeObject(ObjectOutput out, Object element) throws IOException {
            out.writeInt(((Integer)element).intValue());
        }
    }
    class LongSerializer implements SerializerIF {
        public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
```

Figure 110

11000
pg 5

```
                long value = in.readLong();

                return new Long(value);
        }
        public void writeObject(ObjectOutput out, Object element) throws IOException {
                out.writeLong(((Long)element).longValue());
        }
}
class ObjectArraySerializer implements SerializerIF {
        public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
                int size = in.readShort();

                Object[] array = new Object[size];
                for (int i = 0; i < size; i++) {
                        SerializerIF serializer = BaseSerializer.getInstance();
                        array[i] = serializer.readObject(in);
                }

                return array;
        }
        public void writeObject(ObjectOutput out, Object element) throws IOException {
                Object[] array = (Object[])element;

                out.writeShort(array.length);
                for (int i = 0; i < array.length; i++) {
                        SerializerIF serializer = BaseSerializer.getInstance();
                        serializer.writeObject(out, array[i]);
                }
        }
}
class ObjectSerializer implements SerializerIF {
        public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
                return in.readObject();
        }
        public void writeObject(ObjectOutput out, Object element) throws IOException {
                out.writeObject(element);
        }
}
class ShortSerializer implements SerializerIF {
        public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
                short value = in.readShort();

                return new Short(value);
        }
```

Figure 110

11000
pg 6

```
		public void writeObject(ObjectOutput out, Object element) throws IOException {
			out.writeShort(((Short)element).shortValue());
		}
	}
	class SimpleTimeZoneSerializer implements SerializerIF {
		public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
			int offset = in.readInt();
			SerializerIF serializer = BaseSerializer.getInstance();
			String id = (String)serializer.readObject(in);

			return new SimpleTimeZone(offset, id);
		}
		public void writeObject(ObjectOutput out, Object element) throws IOException {
			SimpleTimeZone temp = (SimpleTimeZone)element;

			out.writeInt(temp.getRawOffset());
			SerializerIF serializer = BaseSerializer.getInstance();
			serializer.writeObject(out, temp.getID());
		}
	}
	class StringSerializer implements SerializerIF {
		public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
			int size = in.readShort();
			byte[] bytes = new byte[size];
			in.readFully(bytes);

			return new String(bytes);
		}
		public void writeObject(ObjectOutput out, Object element) throws IOException {
			byte[] bytes = ((String)element).getBytes();

			out.writeShort(bytes.length);
			out.write(bytes);
		}
	}
	class VectorSerializer implements SerializerIF {
		public Object readObject(ObjectInput in) throws ClassNotFoundException, IOException {
			int size = in.readShort();

			Vector vector = new Vector(size);
			for (int i = 0; i < size; i++) {
				SerializerIF serializer = BaseSerializer.getInstance();
				vector.addElement(serializer.readObject(in));
			}
```

Figure 110

11000
pg 7

```
            return vector;
        }
        public void writeObject(ObjectOutput out, Object element) throws IOException {
            Vector temp = (Vector)element;

            Object[] array = new Object[temp.size()];
            for (int i = 0; i < array.length; i++) {
                array[i] = temp.elementAt(i);
            }

            out.writeShort(array.length);
            for (int i = 0; i < array.length; i++) {
                SerializerIF serializer = BaseSerializer.getInstance();
                serializer.writeObject(out, array[i]);
            }
        }
    }
/**
 * Default constructor.  The constructor is private because this is a
 * singleton class.  When the object is constructed, it initializes its
 * tables.
 */
private BaseSerializer() {
    init();
}
/**
 * Adds the given elements to the three tables.
 *
 * @param className the name of the class
 * @param code the code for the given class
 * @param serializer the object responsible for serializing the given
 * class
 */
private void addDataToTables(String className, Number code, SerializerIF serializer) {
    getCodeTable().put(code, className);
    getNameTable().put(className, code);

    if (serializer != null) {
        getSerializerTable().put(code, serializer);
    }
}
```

Figure 110

11000
pg 8

```
/**
 * Creates the codes and serializer objects for the default serialization
 * classes and adds them to the tables. The tables are then written to
 * a serialized file.
 */
private void createDefaultTables() {
        addDataToTables(BigDecimal.class.getName(), new Byte((byte)1), new
BigDecimalSerializer());
        addDataToTables(BigInteger.class.getName(), new Byte((byte)2), new BigIntegerSerializer());
        addDataToTables(Boolean.class.getName(), new Byte((byte)3), new BooleanSerializer());
        addDataToTables(Byte.class.getName(), new Byte((byte)4), new ByteSerializer());
        addDataToTables(Character.class.getName(), new Byte((byte)5), new CharacterSerializer());
        addDataToTables(Date.class.getName(), new Byte((byte)6), new DateSerializer());
        addDataToTables(Double.class.getName(), new Byte((byte)7), new DoubleSerializer());
        addDataToTables(Float.class.getName(), new Byte((byte)8), new FloatSerializer());
        addDataToTables(GregorianCalendar.class.getName(), new Byte((byte)9), new
GregorianCalendarSerializer());
        addDataToTables(Integer.class.getName(), new Byte((byte)10), new IntegerSerializer());
        addDataToTables(Long.class.getName(), new Byte((byte)11), new LongSerializer());
        addDataToTables(Short.class.getName(), new Byte((byte)12), new ShortSerializer());
        addDataToTables(SimpleTimeZone.class.getName(), new Byte((byte)13), new
SimpleTimeZoneSerializer());
        addDataToTables(String.class.getName(), new Byte((byte)14), new StringSerializer());
        addDataToTables(Vector.class.getName(), new Byte((byte)15), new VectorSerializer());
        addDataToTables(Object.class.getName(), new Byte((byte)16), new ObjectSerializer());

        writeTables();
}
/**
 * Returns an instance of the table of class names, keyed by their code.
 * If the table does not exist, it is created.
 *
 * @return The table of class names.
 */
protected Hashtable getCodeTable() {
        if (codeTable == null) {
                codeTable = new Hashtable();
        }
```

Figure 110

11000
pg 9

```
            return codeTable;
    }
    /**
     * Returns an instance of <code>BaseSerializer</code>.
     *
     * @return An instance of <code>BaseSerializer</code>.
     */
    public static SerializerIF getInstance() {
            if (instance == null) {
                    instance = new BaseSerializer();
            }

            return instance;
    }
    /**
     * Returns an instance of the table of codes, keyed by their
     * corresponding class name.
     * If the table does not exist, it is created.
     *
     * @return The table of codes.
     */
    protected Hashtable getNameTable() {
            if (nameTable == null) {
                    nameTable = new Hashtable();
            }

            return nameTable;
    }
    /**
     * Returns an instance of the table of serializers, keyed by their
     * corresponding code.
     * If the table does not exist, it is created.
     *
     * @return The table of class names.
     */
    protected Hashtable getSerializerTable() {
            if (serializerTable == null) {
                    serializerTable = new Hashtable();
            }

            return serializerTable;
    }
    /**
     * Initializes the hashtable from either a serialized hashtable or from
     * an ini file.
     */
```

```
protected void init() {
        File serializedFile = new File(HASHTABLE_SER);
        File iniFile = new File(INI_FILE);

        if (serializedFile.exists()) {
                readSerializedFile(serializedFile);
        } else {
                if (iniFile.exists()) {
                        readIniFile(iniFile);
                }

                createDefaultTables();
        }
}
/**
 * Gets the value of the serialization code from the table based on
 * the className provided.  The value returned can either be a
 * <code>Byte</code> or an <code>Integer</code>.  The return value
 * will be a <code>Byte</code> if the className is one of the base
 * data types.
 *
 * @return The serialization code of the className.
 * @param className the name of the class
 */
private Number lookupCode(String className) {
        Number code = null;

        if (className != null) {
                code = (Number)getNameTable().get(className);
        }

        return code;
}
/**
 * Looks up the hashcode in the table and returns the String value of
 * the hashcode.  If the hashcode does not exist in the table
 * <code>null</code> is returned.
 *
 * @return The object that was stored in the table with the given
 *                 hashcode.
 * @param hashcode the hashcode that will be used to look up the value
 */
```

Figure 110

11000
pg 10

Figure 110

11000
pg 11

```
private String lookupName(Number code) {
        String className = null;

        if (code != null) {
                className = (String)getCodeTable().get(code);
        }

        return className;
}
/**
 * Default method comment.
 *
 * <P>PRE:
 * <P>POST:
 *
 * @return Parameter not modified
 * @return com.ibm.jtc.util.SerializerIF
 * @param code int
 */
private SerializerIF lookupSerializer(Number code) {
        SerializerIF serializer = null;

        if (code != null) {
                serializer = (SerializerIF)getSerializerTable().get(code);
        }

        return serializer;
}
/**
 * Default method comment.
 *
 * <P>PRE:
 * <P>POST:
 *
 * @return Parameter not modified
 * @param iniFile java.io.File
 */
private void readIniFile(File iniFile) {
        BufferedReader in = null;

        try {
                in = new BufferedReader(new FileReader(iniFile));

                for (String inLine = in.readLine(); inLine != null; inLine = in.readLine()) {
                        String trimLine = inLine.trim();
```

```
            if ((trimLine.length() > 0) &&
                !trimLine.startsWith("#")) {
                StringTokenizer tokenizer = new StringTokenizer(trimLine);

                String className = tokenizer.nextToken();
                Integer code = new Integer(className.hashCode());
                SerializerIF serializer = null;

                if (tokenizer.hasMoreTokens()) {
                    String serializerName = tokenizer.nextToken();

                    try {
                        serializer = (SerializerIF)Class.forName(serializerName).newInstance();
                    } catch(Exception e) { }
                }

                addDataToTables(className, code, serializer);
            }
        }
    } catch (Exception throwAway) {
    } finally {
        try {
            in.close();
        } catch (Exception throwAway) {
        }
    }

    writeTables();
}
/**
 * Reads the object from the stream by first reading the code for the
 * element then reads the appropriate data for that object.
 *
 * @return The object that was read from the stream.
 * @param in the input stream that contains the object
 */
public Object readObject(ObjectInput in)
    throws ClassNotFoundException, IOException {
    Object retVal = null;
    Number code = null;

    byte baseCode = in.readByte();
```

Figure 110

11000
pg 12

Figure 110

11000
pg 13

```
        if (baseCode == NULL_OBJECT) {
            retVal = null;
        } else {
            if (baseCode != OTHER) {
                code = new Byte(baseCode);
            } else {
                int secondCode = in.readInt();
                code = new Integer(secondCode);
            }

            SerializerIF serializer = lookupSerializer(code);
            if (serializer != null) {
                retVal = serializer.readObject(in);
            } else {
                String className = lookupName(code);

                try {
                    retVal = Class.forName(className).newInstance();

                    if (retVal instanceof Externalizable) {
                        ((Externalizable)retVal).readExternal(in);
                    } else {
                        retVal = in.readObject();
                    }
                } catch(Exception e) {
                }
            }
        }

        return retVal;
    }
    /**
     * Reads the file containing the serialized hashtables of data.
     *
     * @param serializedFile the file containinig the serialized tables
     */
    private void readSerializedFile(File serializedFile) {
        ObjectInputStream in = null;
        try {
            in = new ObjectInputStream(new FileInputStream(serializedFile));
            codeTable = (Hashtable)in.readObject();
            nameTable = (Hashtable)in.readObject();
            serializerTable = (Hashtable)in.readObject();
        } catch (Exception throwAway) {
        } finally {
                try {
                        in.close();
                } catch (Exception throwAway) { }

                if ((codeTable == null) ||
                        (nameTable == null) ||
                        (serializerTable == null)) {

                        createDefaultTables();
                }
        }
}
/**
 * Writes the given object to the stream. First, the code representing
 * the type of the object is written, then the data within the object
 * is written.
 *
 * @param out the output stream that will contain the object
 * @param element the data object that will be written
 */
public void writeObject(ObjectOutput out, Object element)
        throws IOException {

        if (element == null) {
                out.writeByte(NULL_OBJECT);
        } else {
                String className = element.getClass().getName();
                Number code = lookupCode(className);

                if (code != null) {
                        if (code instanceof Byte) {
                                out.writeByte(code.byteValue());
                        } else if (code instanceof Integer) {
                                out.writeByte(OTHER);
                                out.writeInt(code.intValue());
                        }

                        SerializerIF serializer = lookupSerializer(code);

                        if (serializer != null) {
                                serializer.writeObject(out, element);
                        } else if (element instanceof Externalizable) {
                                ((Externalizable)element).writeExternal(out);
```

Figure 110

11000
pg 14

Figure 110

11000
pg 15

```
                    } else {
                            out.writeObject(element);
                    }
            } else {
                    if (element instanceof Object[]) {
                            className = Object[].class.getName();
                    } else {
                            className = Object.class.getName();
                    }

                    code = lookupCode(className);
                    SerializerIF serializer = lookupSerializer(code);

                    out.writeByte(code.byteValue());
                    serializer.writeObject(out, element);
            }
        }
}
/**
 * Writes the tables to the file.
 */
private void writeTables() {
        ObjectOutputStream out = null;

        try {
                File serFile = new File(HASHTABLE_SER);
                out = new ObjectOutputStream(new FileOutputStream(serFile));

                out.writeObject(getCodeTable());
                out.writeObject(getNameTable());
                out.writeObject(getSerializerTable());
                out.writeObject(new Date());
        } catch(Exception e) {
        } finally {
                try {
                        out.close();
                } catch(Exception e) { }
        }
 }
 }
```

Figure 111

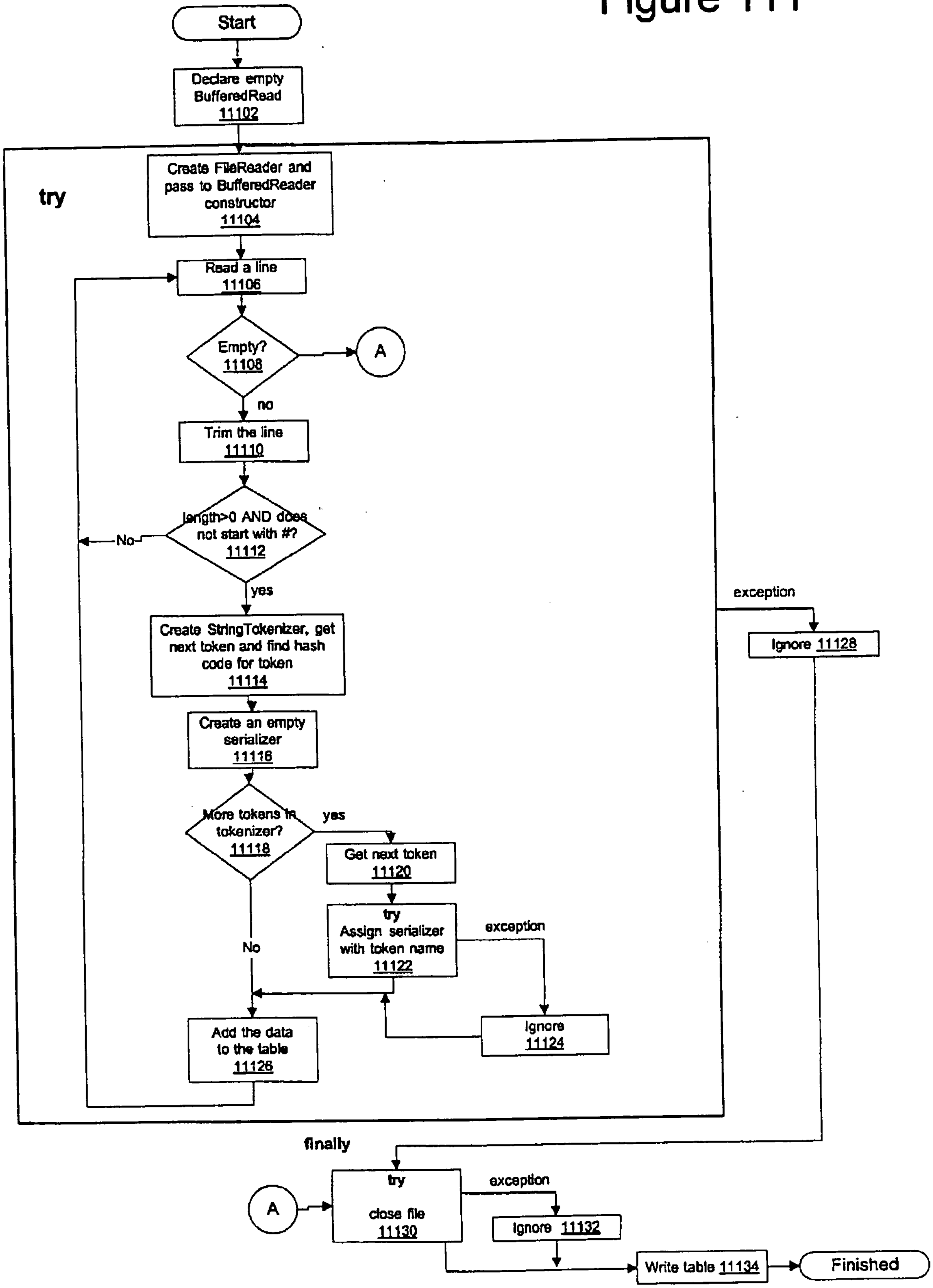

Start
Declare empty BufferedRead 11102
try
Create FileReader and pass to BufferedReader constructor 11104
Read a line 11106
Empty? 11108
A
no
Trim the line 11110
length>0 AND does not start with #? 11112
No
yes
Create StringTokenizer, get next token and find hash code for token 11114
Create an empty serializer 11116
More tokens in tokenizer? 11118
yes
Get next token 11120
try Assign serializer with token name 11122
exception
Ignore 11124
No
Add the data to the table 11126
exception
Ignore 11128
finally
A
try close file 11130
exception
Ignore 11132
Write table 11134
Finished

Figure 112

Getting/Setting
Permissions

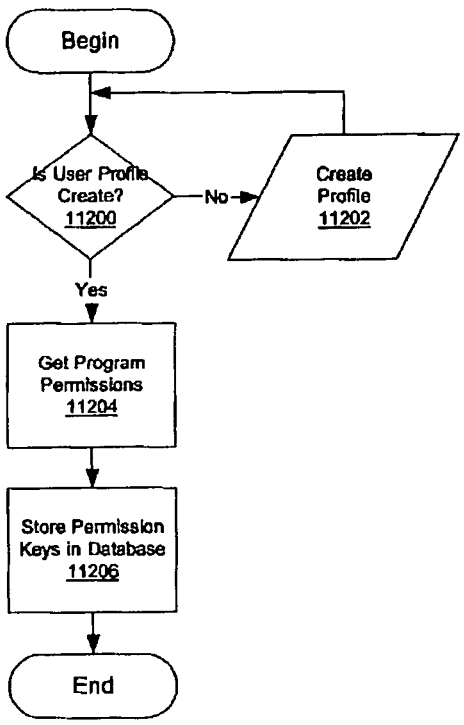


Figure 113

Getting/Setting Permissions

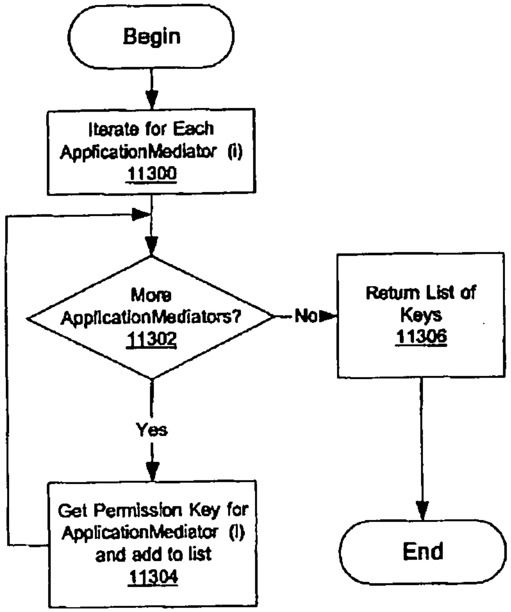

Begin
Iterate for Each ApplicationMediator (i) 11300
More ApplicationMediators? 11302
No
Return List of Keys 11306
End
Yes
Get Permission Key for ApplicationMediator (I) and add to list 11304

Figure 114

Getting/Setting Permissions

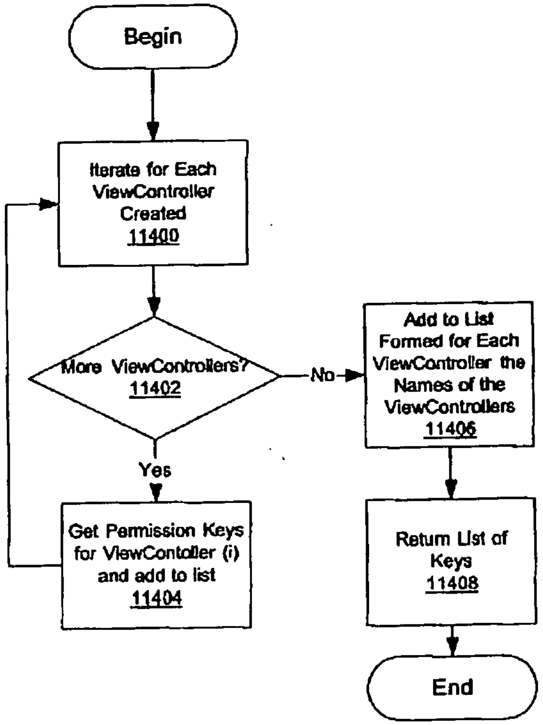

Begin
Iterate for Each ViewController Created 11400
More ViewControllers? 11402
No
Add to List Formed for Each ViewController the Names of the ViewControllers 11406
Return List of Keys 11408
End
Yes
Get Permission Keys for ViewContoller (i) and add to list 11404

Figure 115

Getting/Setting Permissions

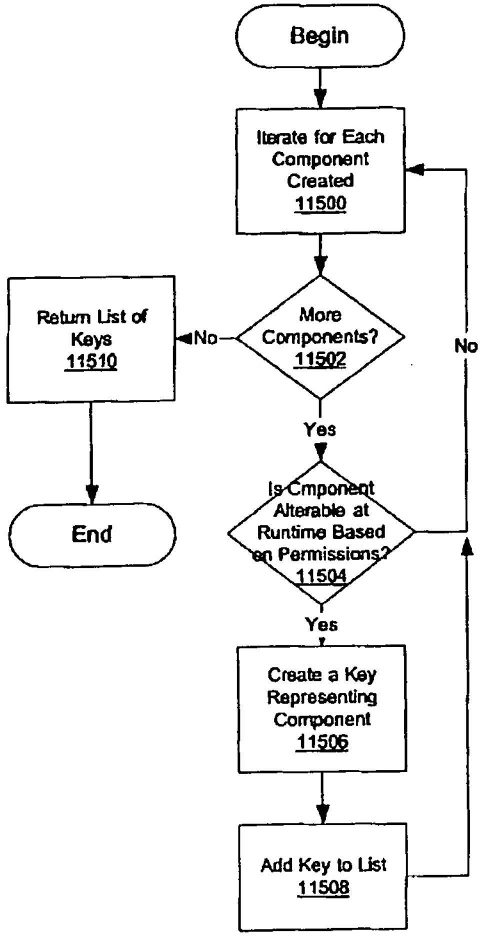

Begin
Iterate for Each Component Created 11500
More Components? 11502
No
Return List of Keys 11510
End
Yes
Is Cmponent Alterable at Runtime Based on Permissions? 11504
No
Yes
Create a Key Representing Component 11506
Add Key to List 11508

Figure 116

Getting/Setting Permissions

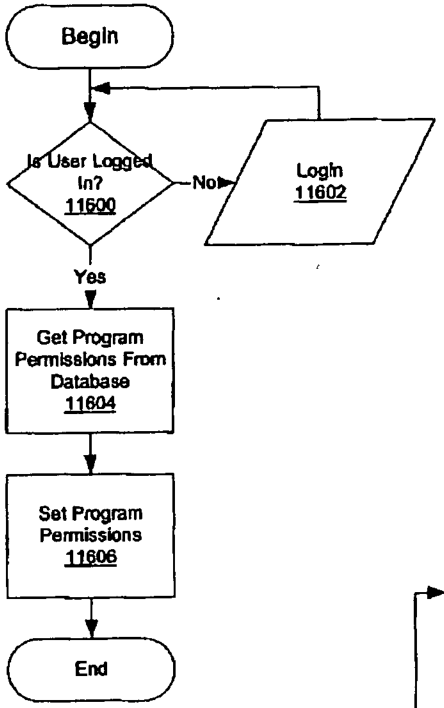

Begin
Is User Logged In? 11600
No
Login 11602
Yes
Get Program Permissions From Database 11604
Set Program Permissions 11606
End

Figure 117

Getting/Setting Permissions

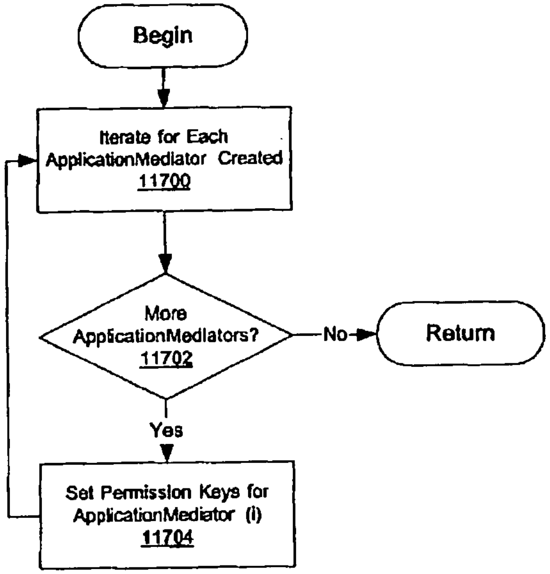

Begin
Iterate for Each ApplicationMediator Created 11700
More ApplicationMediators? 11702
No
Return
Yes
Set Permission Keys for ApplicationMediator (I) 11704

Figure 119
Getting/Setting Permissions

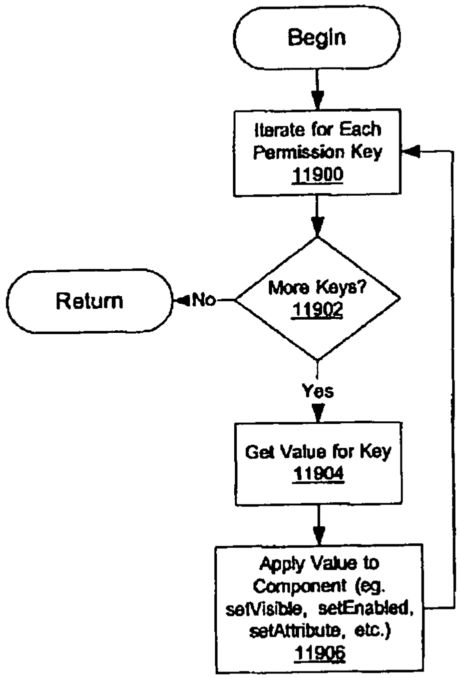
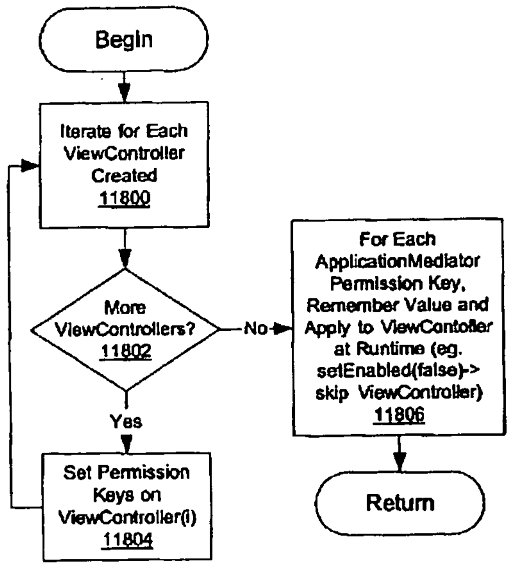
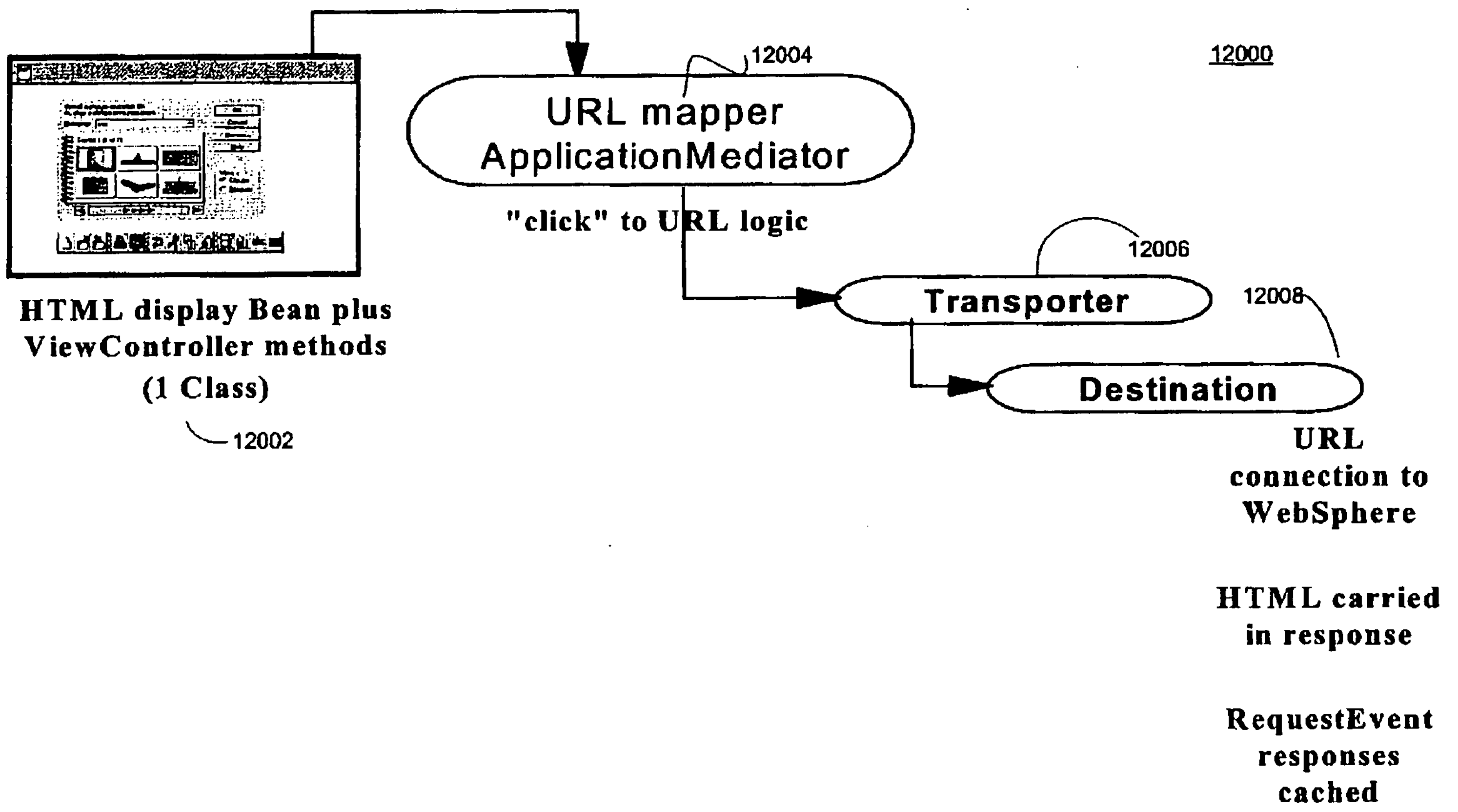

Begin
Iterate for Each Permission Key 11900
More Keys? 11902
No
Return
Yes
Get Value for Key 11904
Apply Value to Component (eg. setVisible, setEnabled, setAttribute, etc.) 11906

Figure 118
Getting/Setting Permissions


Begin
Iterate for Each ViewController Created 11800
More ViewControllers? 11802
Yes
Set Permission Keys on ViewController(i) 11804
No
For Each ApplicationMediator Permission Key, Remember Value and Apply to ViewContoller at Runtime (eg. setEnabled(false)-> skip ViewController) 11806
Return

Example Pattern FIGURE 120

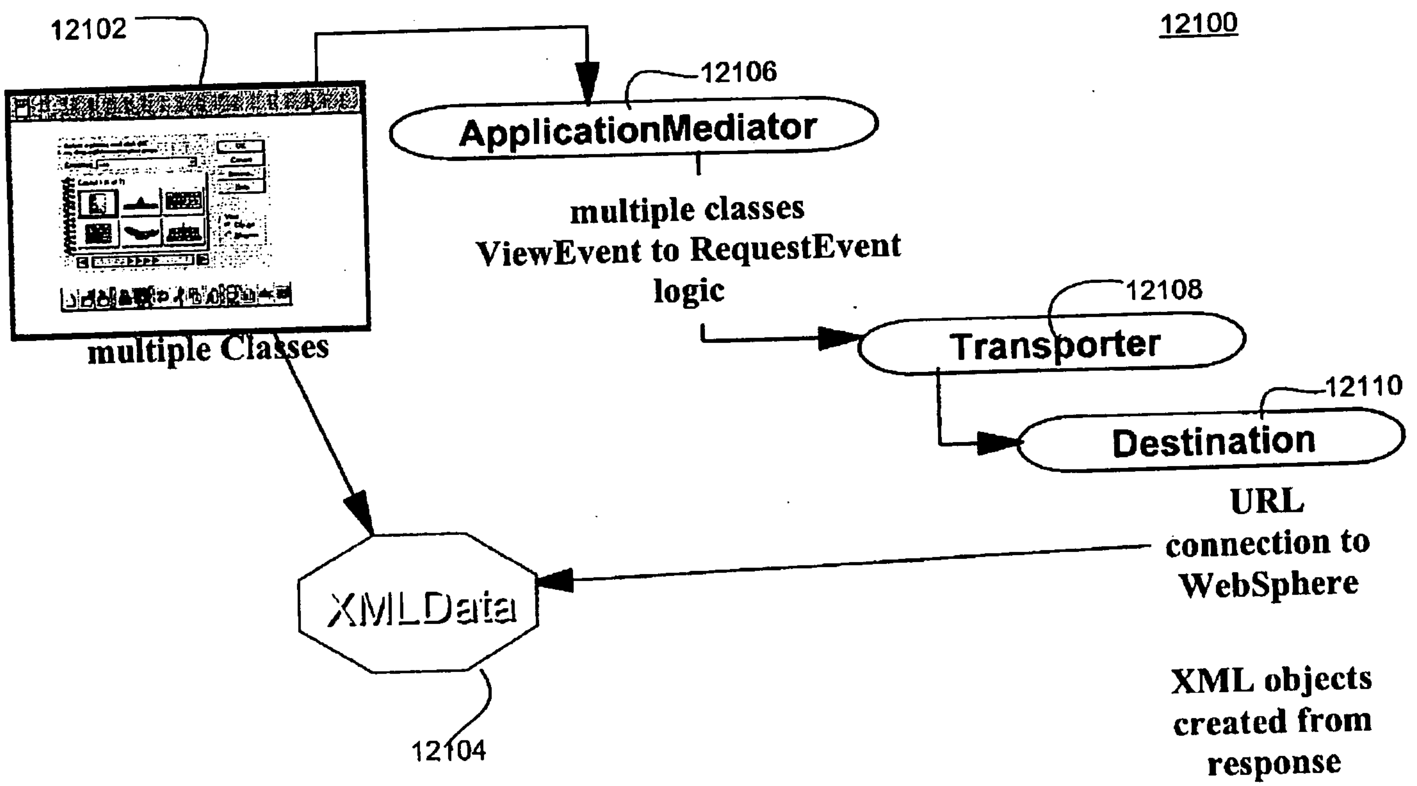
- ➡ Interpreted virtual screens
- ➡ Transactional with caching


12000
URL mapper ApplicationMediator
12004
"click" to URL logic
Transporter
12006
Destination
12008
HTML display Bean plus ViewController methods (1 Class)
12002
URL connection to WebSphere
HTML carried in response
RequestEvent responses cached

Example Pattern

FIGURE 121

- ➡ Data models and ViewControllers
- ➡ Transactional



multiple Classes

multiple classes
ViewEvent to RequestEvent
logic

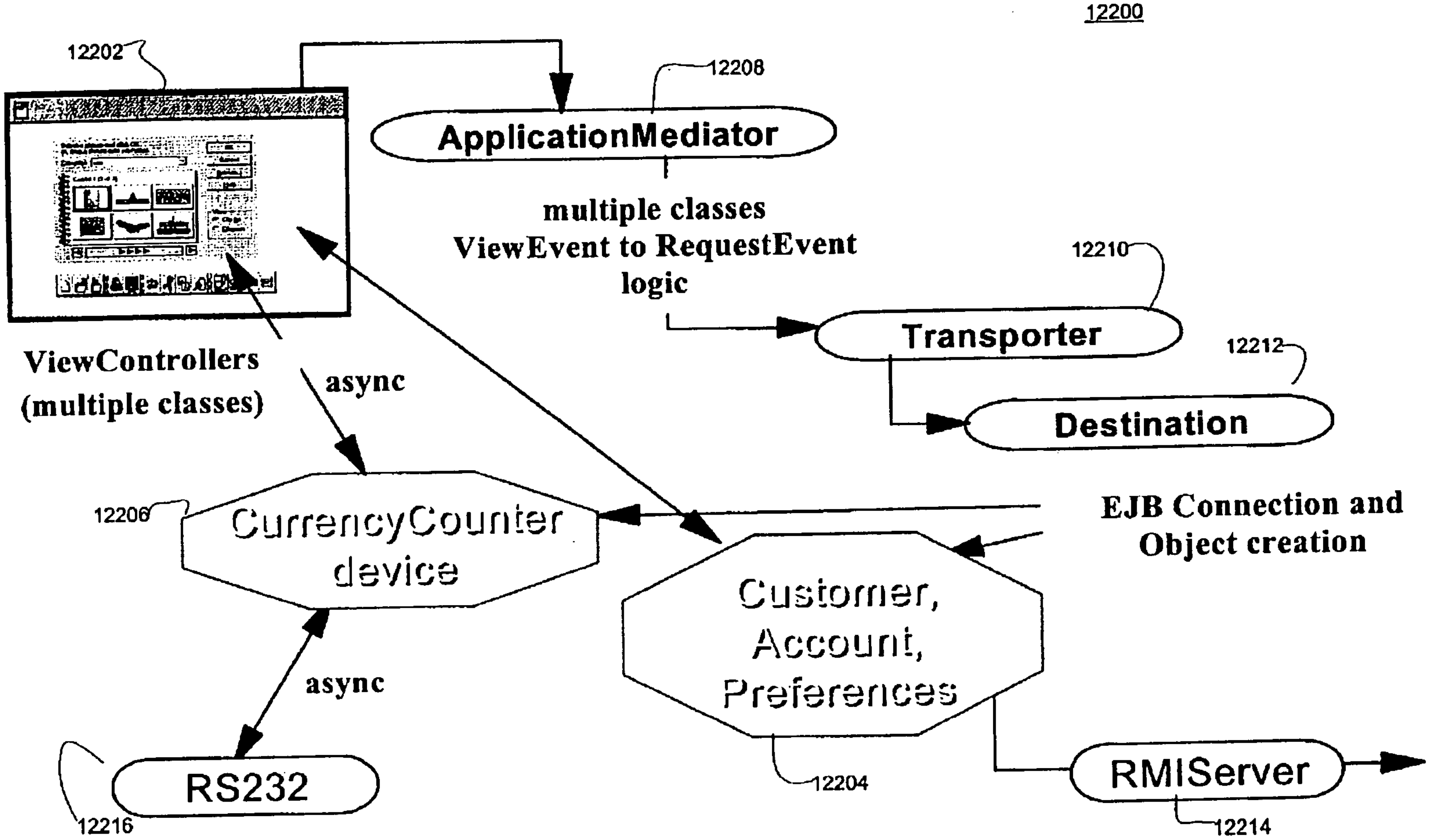
URL
connection to
WebSphere

XML objects
created from
response

Example Pattern

FIGURE 122

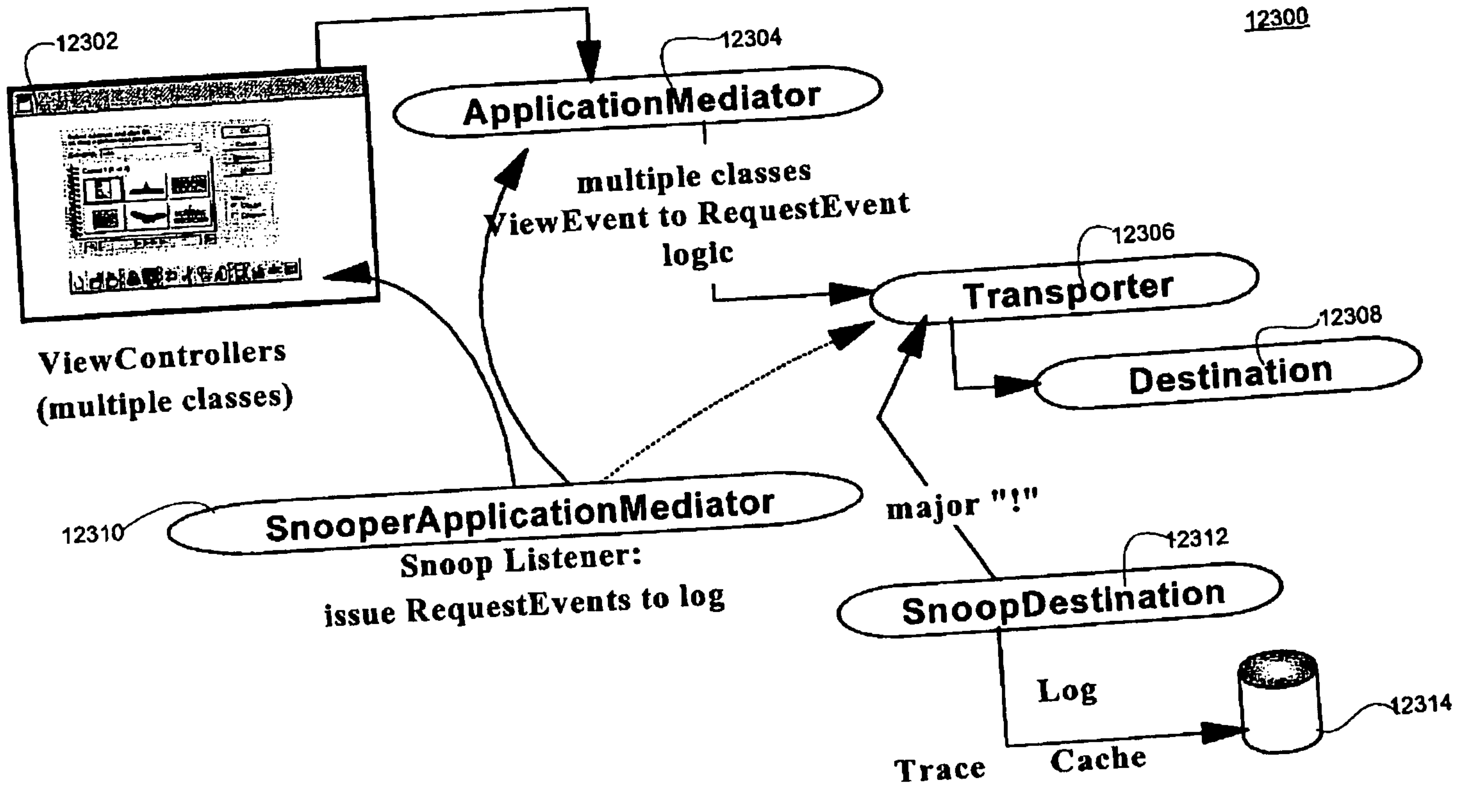
- ➡ Live data objects
- ➡ Streaming and remote objects
- ➡ RequestEvents to turn on/off data objects



Example Pattern

FIGURE 123

- Non-intrusive Caching, Tracing or Logging



12300
12302
12304
ApplicationMediator
multiple classes
ViewEvent to RequestEvent
logic
12306
Transporter
12308
Destination
ViewControllers
(multiple classes)
12310
SnooperApplicationMediator
Snoop Listener:
issue RequestEvents to log
major "!"
12312
SnoopDestination
Log
12314
Trace
Cache